MacTech Magazine
Vol. 16, No. 3 • March 2000

Inside: Large Format Printing

MacTech®

The Journal of Macintosh Technology & Development

C.Y.A.

Backup Management

by Paul Shields

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Also in this issue:

Control Freaks Alert!
by Tom Djajadiningrat

Opening the Toolbox
by Tim Monroe

EtherPeek Review
by Bill von Hagen

Quickly Build and Deploy Powerful Data-Driven Web Applications

Lasso Studio and Lasso Web Data Engine™ Lead The Way.

Building custom and shrink-wrapped database-driven Web applications requires a whole new way of doing things. Having pioneered the Web Data Engine™ over three years ago, Blue World and the Lasso Web Data Engine consistently lead the way providing a feature set Web developers describe as "incredible." Build online stores, discussion forums, resource management systems and other demanding database-driven Web applications with unrivaled performance, ease, security, extensibility, control and flexibility. Develop using multiple languages—including LDML, CDML, Server-Side JavaScript, Java and XML— and deploy across multiple platforms. Your Lasso code works identically regardless to which database you're connected. Lasso solutions for FileMaker® Pro databases easily scale to big iron ODBC-compliant databases like Oracle, Informix, Sybase and more with little or no change. What's more, the Lasso Java Application Programming Interface (LJAPI) provides developers an easy-to-use Java-based API for unprecedented extensibility.

Find out why hundreds of thousands of websites rely on award-winning Lasso technology for their business critical Web data. Download a 30-day evaluation copy at www.blueworld.com/download/ or order securely online today at the Blue World Store at store.blueworld.com.

Lasso Product Line – The leading Web tools for Macintosh and beyond.

bring business to the internet™

blueworld

"Without a doubt, the Premiere Resource Editor for the Mac OS ... A wealth of time-saving tools."
– MacUser Magazine Eddy Awards

"A distinct improvement over Apple's ResEdit."
– MacTech Magazine

"Every Mac OS developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Without Resorcerer, our localization efforts would look like a Tower of Babel. Don't do product without it!"
– Greg Galanos, CEO and President, Metrowerks

"Resorcerer's data template system is amazing."
– Bill Goodman, author of Smaller Installer and Compact Pro

"Resorcerer Rocks! Buy it, you will NOT regret it."
– Joe Zobkiw, author of ***A Fragment of Your Imagination***

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

Resorcerer® 2

Version 2.0

The Resource Editor for the Mac™ OS Wizard

ORDERING INFO

Requires System 7.0 or greater,
1.5MB RAM, CD-ROM

Standard price: $256 (decimal)
Website price: $128 - $256
(Educational, quantity, or
other discounts available)

Includes: Electronic documentation
60-day Money-Back Guarantee
Domestic standard shipping

Payment: Check, PO's, or Visa/MC
Taxes: Colorado customers only

Extras (call, fax, or email us):
COD, FedEx, UPS Blue/Red,
International Shipping

MATHEMÆSTHETICS, INC.
PO Box 298
Boulder, CO 80306-0298 USA
Phone: (303) 440-0707
Fax: (303) 440-0504
resorcerer@mathemaesthetics.com

New in 2.0:

- **Very fast, HFS browser for viewing file tree of all volumes**
- **Extensibility for new Resorcerer Apprentices (CFM plug-ins)**
- **New AppleScript Dictionary ('aete') Apprentice Editor**
- **MacOS 8 Appearance Manager-savvy Control Editor**
- **PowerPlant text traits and menu command support**
- **Complete AIFF sound file disassembly template**
- **Big-, little-, and even mixed-endian template parsing**
- **Auto-backup during file saves; folder attribute editing**
- **Ships with PowerPC native, fat, and 68K versions**

- Fully supported; it's easier, faster, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Balloons, AppleEvent, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Relied on by thousands of Macintosh developers around the world

To order by credit card, or to get the latest news, bug fixes, updates, and apprentices, visit our website...

www.mathemaesthetics.com

MacTech®
The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin
Managing Editor • Jessica Stubblefield
Editor • Nick DeMello
Online Editor • Jeff Clites
Web Editor • Kevin Avila

Regular Columnists

Getting Started
by Dan Parks Sydow
Programmer's Challenge
by Bob Boonstra
MacTech Online
by Jeff Clites
From the Factory Floor
by Richard Atwell, Metrowerks
Tips & Tidbits
by Jeff Clites

Regular Contributors

Vicki Brown, Erick Tejkowski, Kas Thomas, Will Porter, Paul E. Sevinç, and John C. Welch

MacTech's Board of Advisors

Dave Mark, Jordan Dea-Mattson, Jim Straus, Jon Wiederspan, and Eric Gundrum

PRINTED WITH SOY INK

This publication is printed on paper with recycled content.

MacTech's Contributing Editors

- Jim Black, Apple Computer, Inc.
- Michael Brian Bentley
- Tantek Çelik, Microsoft Corporation
- Marshall Clow, Adobe
- John. C. Daub
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Cobalt Networks
- John Hanay, Apple Computer
- Lorca Hanns, San Francisco State University
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Scott Knaster, Microsoft
- Mark Kriegsman, Clearway Technologies
- Peter N. Lewis, Stairways Software
- Bill McGlasson, Apple Computer
- Rich Morin, PrimeTime Freeware, Inc.
- Terje Norderhaug, Media*Design-In-Progress
- Nathan Nunn, Purity Software
- John O'Fallon, Maxum Development
- Alan Oppenheimer, Open Door Networks
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Dori Smith
- Michael Swan, Neon Software
- Chuck Von Rospach, Plaidworks
- Bill von Hagen
- Eric Zelenka, StarNine

XPLAIN CORPORATION

Chief Executive Officer • Neil Ticktin
President • Andrea J. Sniderman
Controller • Michael Friedman
Production Manager • Jessica Stubblefield
Production • Terrell Dunn
Marketing Manager • Nick DeMello
Events Manager • Susan M. Worley
Network Administrator • Steve Ruge
Accounting • Jan Webber, Marcie Moriarty
Customer Relations • Laura Lane
Susan Pomrantz
Shipping/Receiving • Joel Licardie

Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

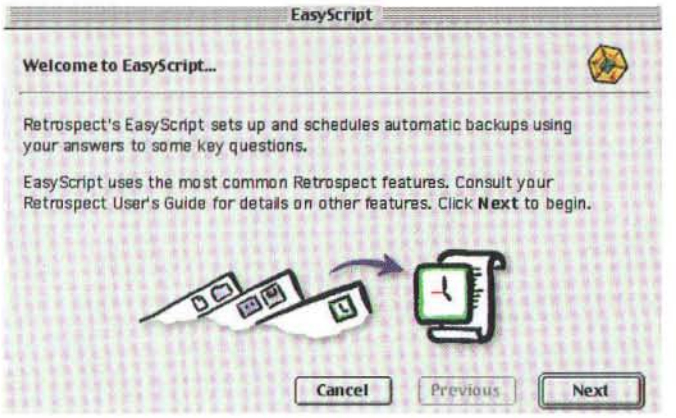
All contents are Copyright 1984-2000 by Xplain Corporation. All rights reserved. MacTech, Developer Depot, and Sprocket are registered trademarks of Xplain Corporation. Depot, The Depot, Depot Store, Video Depot, MacDev-1, TIIINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, Explainlt, and the MacTutorMan are trademarks of Xplain Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders. Xplain Corporation does not assume any liability for errors or omissions by any of the advertisers, in advertising content, editorial, or other content found herein. Opinions or expressions stated herein are not necessarily those of the publisher and therefore, publisher assumes no liability in regards to said statements.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

March 2000 • Volume 16, Issue 03



Backup Management .....page 36

Feature Articles

DATA ACQUISITION ON THE MACINTOSH

20 **Control Freaks Alert!**
by Tom Djajadiningrat

SYSTEMS

36 **Backup Management**
by Paul Shields

TOOLS OF THE TRADE

14 **Put Your Network Under a Microscope with EtherPeek**
by Bill von Hagen

90 **Large Format Printing**
by MacTech Production Staff

PROGRAMMING

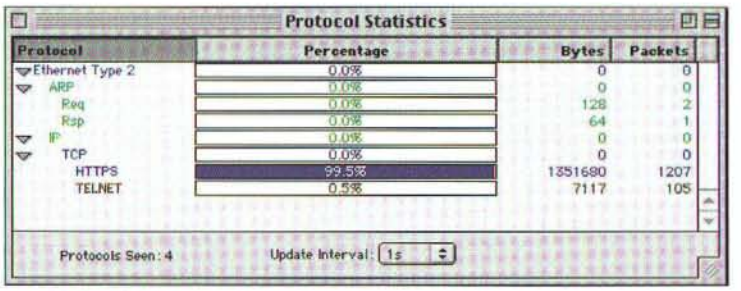
44 **Not Your Father's Resource Template**
by Neil Mayhew

51 **Dynamic PDF Made Easy**
by Kas Thomas

QUICKTIME TOOLKIT

60 **Opening the Toolbox**
by Tim Monroe

Opening the Toolbox.....page 60



EtherPeek Review.....page 14

Columns

VIEWPOINT

4 **Macworld SF 2000 Show Report**
by Nick DeMello

GETTING STARTED

6 **Speech and Voices**
by Dan Parks Sydow

PROGRAMMER'S CHALLENGE

74 **Sum of Powers**
by Bob Boonstra

FROM THE FACTORY FLOOR

92 **CodeWarrior Update**
by Richard Atwell

Monthly Features

94 MacTech Online *by Jeff Clites*

96 Advertiser & Product Index

VIEWPOINT

By Nick DeMello

This May, Apple is hosting it's Worldwide Developers Conference in San Jose California. Promoting development solutions for Macintosh is essential to Apple's survival. Apple makes their money by selling Macintosh, but people only buy Macintosh to make use of those solutions. Apple isn't in the business of creating and selling word processing programs, browsers, compilers, games, tools like Photoshop, Premiere, Eudora, FileMaker, Excel and the next great and unknown product that some is building in his basement right now. But Apple wouldn't be able to sell very many Macs if those solutions didn't exist and weren't continuing to be developed for Macintosh. WWDC is the one week a year when Apple calls together as many developers as they can and makes their pitch as to why people should develop for Macintosh.

The convention will be broken into at least five tracks of related presentations and workshops. The first of these tracks will focus on the Mac OS. This core OS is one of the most important parts of Macintosh. This year the focus will be on Mac OS X. Expect presentations on just about every part of Mac OS X: the Darwin Core, Quartz imaging model, Aqua interface, Cocoa application framework, and Carbon compliance for migrating older applications. For years Apple has taken heat about the core of their system lacking such essential features a true multitasking and protected memory, this year they'll finally be able to deliver those as well as some truly innovative and fundamental new technologies that will provide some compelling reasons for developers to target Macintosh.

Track two is a digital media track that will include QuickTime, OpenGL, and (not sure why this one is in Digital Media) Java. Over the last couple of years, Apple has extended their definition of "developer" in the same way the entire industry has. At one point providing solutions on the Macintosh meant you were a programmer, but today content creators and media developers are a growing portion of the folks bringing products to the Macintosh platform. DVD movies and Streaming Media Channels are almost as compelling a reason to buy Macintosh as traditional software. Expect to see sessions focused on preparing, manipulating, and delivering rich digital media with these technologies. The message in this track is that Macintosh is the definitive platform for digitial media content and application development and delivery.

Apple is a hardware company. The hardware track promises to give us a developers perspective on Apple's current hardware technologies (including AltiVec and AirPort), interface standards like FireWire and USB, and a look behind the curtain at what's in the works for the next year. Apple will make arguments for why the hardware component of Macintosh gives it a substantial edge over other platforms now and in the next year. They will provide reasons to locate your cpu or graphics intensive applications on Macintosh, as well try and outline opportunities to develop third party accessories to expand and support Macintosh. The explosion of USB devices and potential of FireWire based products for the Macintosh is pretty good demonstration of the effectiveness of these arguments from last year.

The Network and Security track gives some insight into how Apple sees the Macintosh platform supporting a new generation of applications. Apple's Sherlock 2, QuickTime TV, iDisk, and KidSafe are showcases of how the new Mac OS can allow desktop applications to work transparently with internet resources. The powerful new security, multi-user, and encryption features of the Mac OS coupled with new kinds of connectivity through the Sherlock interface, Streaming QuickTime, AppleShare IP (and Client SDK), NSL, AppleScript TCP/IP, and others mean Macintosh hosted applications can offer some great functionality for e-commerce, communications, and network based computing. While many of these technologies are not new, developer's haven't brought many related applications to market yet. Apple's promotion of iTools and now a WWDC track dedicated to Network and Security seems to indicate their intent jump start development in this area and stand behind the technologies they've developed for it. Personally, I think these technologies are likely to result in some of the most exciting new products for Macintosh and am looking forward to these sessions.

But it's the last track that really caught my eye: Tools. Apple has apparently devoted an entire conference track to developer resources where they will reveal "Apple's tools strategy and the future of development tools on Mac OS." On one hand Apple doesn't have a very good track record with regard to developer resources, I have to look at this announcement through skepticism colored by Bedrock, a lack of any printed API documentation, and charging developers $195 a pop to talk to Apple. On the other hand, the last few years have seen Apple make some bold and effective moves with regard to product design, development schedules, market position, advertising, and distribution. I'm hoping we're going to see a well thought out and long term developer resources plan that is just as innovative and successful. I'm hoping Apple plan's to make building Macintosh applications as easy as they've made using Macintosh and that the fifth track will be the most compelling reason yet to develop for Macintosh. MT

The key to thinking different...

Works on the iMac!

The New MacHASP USB Key!

Question: Is MacHASP USB* a software security key or a sales tool?

Answer: It's both!

MacHASP USB is the world's first software protection key for the iMac. It gives you sophisticated license enforcement and comprehensive protection against illegal use ... in other words, real security.

Then it gives you a big selling advantage.

With MacHASP USB, you can license multiple software modules and applications. You can instantly unlock or upgrade them in the field. Plus you can freely distribute demos.

Bottom line: MacHASP USB locks out illegal users and unlocks your full sales potential – without getting in anyone's way. Call now to request a Developer's Kit and our newly published guide to USB features and benefits.

*For all USB-equipped Macs running an OS with USB support. Fully compatible with the ADB version of MacHASP.

Mac OS

Accelerated for Power Macintosh

USA: 1-800-223-4277
212-564-5678
Int'l: 972-3-6362222

www.aks.com/mt

ALADDIN®
KNOWLEDGE SYSTEMS LTD

Mac software protection provider, Micro Macro Technologies, becomes part of Aladdin, giving its customers even better service from the number one name.

By Dan Parks Sydow

Speech and Voices

Writing a Macintosh program that makes use of different voices

Two months ago, *Getting Started* introduced the topic of adding computer-generated speech to a Mac program. In that article you learned how to verify that the user's Mac is able to output speech and then use the Toolbox SpeakString() function to speak the text of one string. Last month's *Getting Started* article carried on with the topic of speech by demonstrating how a program can create a speech channel in order to speak more than a single string. This month we complete the trilogy of speech articles by seeing how a speech channel is used by a program that wants to alter the voice that's used in the generation of speech.

Speech Basics Review

A Mac program that is to speak should include the Speech.h universal header file to make sure that the compiler recognizes the speech-related Toolbox functions. The program should also verify that the user's Mac is able to generate speech. After that, a string of text can be spoken by calling the Toolbox function SpeakString(). A Pascal-formatted string is the only argument. After that, repeatedly call the Toolbox function SpeechBusy() to allow for the Mac to complete talking. If the following snippet doesn't look familiar, refer back to the January *Getting Started* article for more information.

```
#include <Speech.h>

OSErr  err;
long   response;
long   mask;

err = Gestalt( gestaltSpeechAttr, &response );
if ( err != noErr )
  DoError( "\pError calling Gestalt" );

mask = 1 << gestaltSpeechMgrPresent;
if ( response & mask == 0 )
  DoError( "\pSpeech Manager not present " );

err = SpeakString( "\pThis is a test." );
if ( err != noErr )
  DoError( "\pError attempting to speak a phrase" );

while ( SpeechBusy() == true )
  ;
```

Speech Channels Review

Last month's *Getting Started* introduced speech channels. In order to specify which voice to use, you're program needs to allocate such a channel — so take a quick look at some of the code from last month.

A speech channel is a data structure that holds descriptive information about the voice that is to be used to speak. If you're program needs to alternate voices, you may want to create two or more speech channels — one for each voice. The Toolbox function NewSpeechChannel() creates a new speech channel record and returns a SpeechChannel — a pointer to that record.

```
SpeechChannel channel;
OSErr         err;

err = NewSpeechChannel( nil, &channel );
```

The first NewSpeechChannel() argument is a pointer to a voice specification data structure. Passing a value of nil as the first argument results in assigning the system default voice as the voice to be used by speech that comes eventually comes from this new channel.

Voices

To this point we've relied on the default voice — the voice the user's Mac uses when a program doesn't specify a particular voice. As shown in **Figure 1**, the_user uses the Speech control panel to determine which voice is to be the default voice.

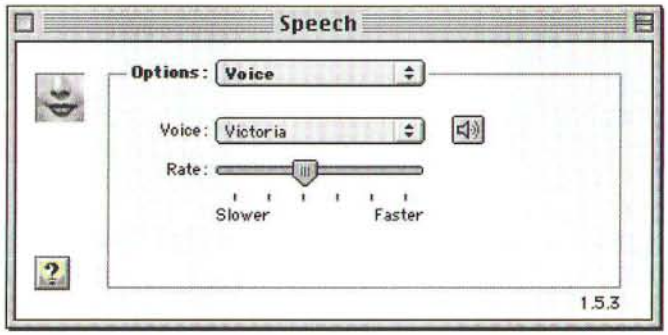
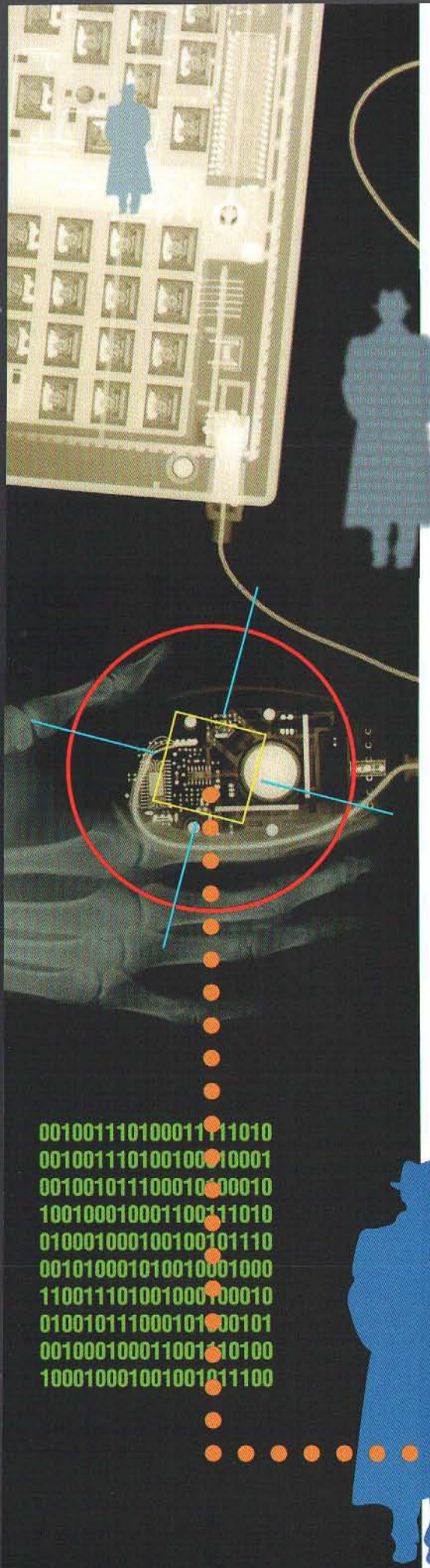


Figure 1. *The Speech control panel.*

As shown on the left of **Figure 2**, the Speech control panel includes a pop-up menu that lets the user choose a voice. And as shown on the right of **Figure 2**, each voice in the Speech control panel pop-up menu corresponds to a voice in the System Folder of the user's Mac (in the Voices folder in the Extensions folder, to be exact).



SPI™ tracks down software pirates with every illegal click.

SPI adds a protective layer to your program that allows only those end-users who legitimately purchase your software to install it. No matter how many copies of your program get distributed illegally, only the valid purchaser can install and run your product. For just pennies a copy, you can protect all your products and significantly increase your revenue without increasing production costs.

Features:

- Win95, Win98, Win2000, WinNT, Mac/PowerMac
- Resource Encryption
- Key File Encryption
- 3 Built-in Demo Modes
- Built-in Online Registration
- Built-in General Purpose Registers
- Distribute via CD/DVD, Diskette, Internet

Sell your software without a shopping cart.

With SPI's built-in e-Commerce feature, distribute and sell your software in one easy step. Your customers simply download your SPI protected product and select Purchase when they run the installer. After entering their credit card information, SPI uses identity authentication, packet encryption and a secure link between SPI and the authorizing agent to protect the transaction. Upon authorization, your product is installed and ready to run. You can even set up a "try-before-you-buy" demo that can then be upgraded using SPI's e-Commerce feature when the demo expires.

Features:

- Real-Time Authorization thru CyberCash™ or Authorize.Net™
- No Web Store Front Required
- 3-tier Transaction Protection
- Upgrade Demos to Full Version from Original Product Installer Download
- Purchase/Tracking Reports Provide Additional Credit Card Charge-Back Protection
- Reduces Web Hosting Charges

No special hardware required.

SPI doesn't require expensive hardware keys or dongles to protect your software. Besides, you can't download a dongle. SPI uses an encrypted software lock to protect installed files. Once a legal product is installed, SPI's Lock Point feature prevents illegal copies from running on any other computer, while SPI's uninstall feature allows end-users to move a protected product to another computer. Simply run the installer again and SPI disables the first installation, allowing the software to be installed on another computer.

See for yourself how SPI protects your products.

Visit our web site at www.nwspi.com to request our SPI Evaluation Kit. The SPI Evaluation Kit will allow you to protect one of your software products for up to one month. Each Evaluation Kit comes complete with the cross-platform SPI software, APIs and the SPI User's Guide outlining all of SPI's features. You can test every feature of SPI, including the built-in e-Commerce feature with a special merchant account set up just for testing purposes. Request your SPI Evaluation Kit today.

SPI™

Software Piracy Intervention *www.nwspi.com*

New Wave Software, Inc. *800-920-9283 info@nwspi.com*

Copyright © 1999 New Wave Software, Inc. All Rights Reserved. Pat. Pend. SPI is a trademark of New Wave Software, Inc. CyberCash is a trademark of CyberCash, Inc. Authorize.Net is a trademark of Authorize.Net Corporation.

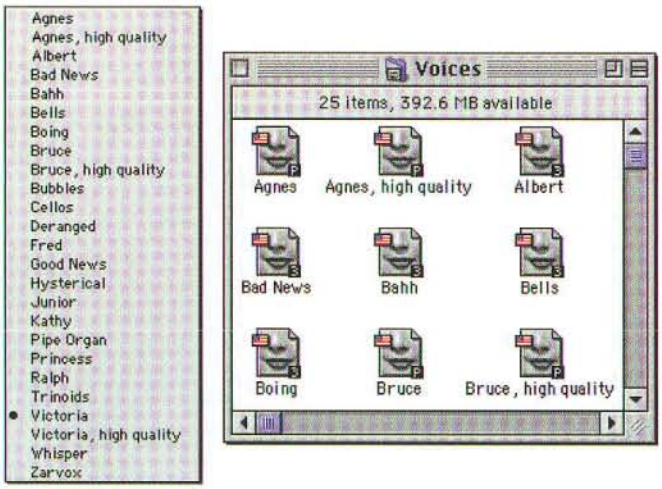


Figure 2. *Each voice in the Speech control panel pop-up menu corresponds to a system voice file.*

As shown in **Figure 2**, voices each have a name. So it might seem logical that your program would choose a voice based on its name. However, such is not the case. Voice files can be added and removed from the Voices folder, and there's no way of ensuring that any one particular voice is present in the Voices folder of every Mac user. So rather than specifying a voice by name, your program should specify one or more characteristics the desired voice should have. After that your program can examine the voices present on the user's machine in order to find an appropriate voice. When a match is found, the voice can be associated with a speech channel, and subsequent text that is sent to that channel will be spoken using the desired voice.

The data that makes up any one voice can be placed in a VoiceDescription data structure. Such a structure has several fields, the most important of which are the gender and age fields. We'll look at these fields after first learning how a VoiceDescription is obtained.

Obtaining the Description of a Voice

To find a voice that meets your needs, your program will examine the voices in the Voices folder of the user's machine. Begin this chore by determining how many voices are in that folder. A call to the Toolbox function CountVoices() returns this number in its one argument. Use the number of voices as a loop counter, where the body of the loop examines each voice for a suitable match.

```
OSErr  err;
short  numVoices;
short  i;

err = CountVoices( &numVoices);

for ( i = 1; i <= numVoices; i++ )
{
  // obtain a voice description for one voice
}
```

Obtaining a voice takes two steps. The first step is to get a VoiceSpec for a voice. The VoiceSpec consists of an identification number of the speech synthesizer for which the voice was created, and an identification number for the voice itself. Any number of voices can share the same speech synthesizer ID, but each voice of those voices with the same speech synthesizer ID will have a voice ID that is unique. Passing the Toolbox function GetIndVoice() an index of a voice in the Voices folder results in GetIndVoice() returning a VoiceSpec for that one voice. We'll be placing the call to GetIndVoice() within the above for loop, so we can use the current loop counter as the index:

```
VoiceSpec theVoiceSpec;

err = GetIndVoice( i, &theVoiceSpec );
```

The VoiceSpec for a voice let's use access the VoiceDescription structure of that voice. Call the Toolbox function GetVoiceDescription() to do that.

```
VoiceDescription voiceDesc;

err = GetVoiceDescription( theVoiceSpec, &voiceDesc,
                           sizeof( voiceDesc ) );
```

The first argument to GetVoiceDescription() is the just-obtained VoiceSpec. The second argument is the VoiceDescription structure, and will be filled in by GetVoiceDescription(). The third argument is the number of bytes in the returned VoiceDescription structure. Use the sizeof() function to get this value. Here's how the voice-selecting code looks — so far:

```
OSErr             err;
short             numVoices;
short             i;
VoiceSpec         theVoiceSpec;
VoiceDescription  voiceDesc;

err = CountVoices( &numVoices);

for ( i = 1; i <= numVoices; i++ )
{
  err = GetIndVoice( i, &theVoiceSpec );

  err = GetVoiceDescription( theVoiceSpec, &voiceDesc,
                             sizeof( voiceDesc ) );

  // examine the characteristics of this one voice
}
```

Selecting a Voice Based On Characteristics

At this point we know how to determine the number of voices in the user's Voices folder and then get a VoiceDescription for each voice. Now we need to look at how we select a voice based on information in that VoiceDescription structure. For most programmers the most meaningful fields of this structure are the gender field and the age field.

The gender field can have one of three values, each represented by an Apple-defined constant. The kMale and kFemale gender values are self-explanatory. The third gender value, kNeuter, describes a voice that is robotic-sounding. The age field holds the approximate age that a speaker of a voice would have.

If you want your program to generate speech using a male voice, you'll be testing the gender field of a VoiceDescription structure. Assuming we've used the above method to obtain a VoiceDescription for a voice, that test looks like this:

```
if ( voiceDesc.gender == kMale )
```

If you want your program to generate speech using a voice of a teenager, then you'd test the age field of a VoiceDescription structure. Again assuming a VoiceDescription has already been

obtained, the code to accomplish the task at hand looks like this:

```
if ( ( voiceDesc.age > 12 ) && ( voiceDesc.age < 20 ) )
  // we've found a teenage voice
```

And if we want to specify a voice based on both the gender and the age? Combine both tests (which order you perform the tests in isn't important).

```
if ( ( voiceDesc.age > 12 ) && ( voiceDesc.age < 20 ) )
{
  if ( voiceDesc.gender == kMale )
    // we've found the voice of a teenage male
}
```

Using the Selected Voice

When we've found VoiceDescription information that matches our voice interests, we've found a suitable voice. If working from within a loop, now's the time to terminate the loop and make use of the VoiceSpec.

```
for ( i = 1; i <= numVoices; i++ )
{
  err = GetIndVoice( i, &theVoiceSpec );

  err = GetVoiceDescription( theVoiceSpec, &voiceDesc,
                             sizeof( voiceDesc ) );

  if ( ( voiceDesc.age > 12 ) && ( voiceDesc.age < 20 ) )
  {
    if ( voiceDesc.gender == kMale )
      // use the VoiceSpec held in variable theVoiceSpec
  }
}
```

Last month you saw the creation of a new speech channel handled in this manner:

```
SpeechChannel channel;

err = NewSpeechChannel( nil, &channel );
```

Using nil as the first argument meant the returned speech channel makes use of the default voice. If we now instead pass the just-obtained VoiceSpec, the speech channel will be created such that it makes use of the voice referenced by this VoiceSpec:

```
err = NewSpeechChannel( &theVoiceSpec, &channel );
```

Last month's *Getting Started* article introduced SpeakText(), the Toolbox function that speaks text from a buffer. The first argument to that routine is a SpeechChannel. Now that the channel variable references a speech channel that's associated with a specific voice, that voice (rather than the system default voice) will be used in the generation of the speech that SpeakText() emits.

```
err = SpeakText( channel, (Ptr)(str + 1), str[0] );
```

SpeechVoice

This month's program is SpeechVoice. When you run SpeechVoice you'll be presented with the dialog box pictured in **Figure 3**.

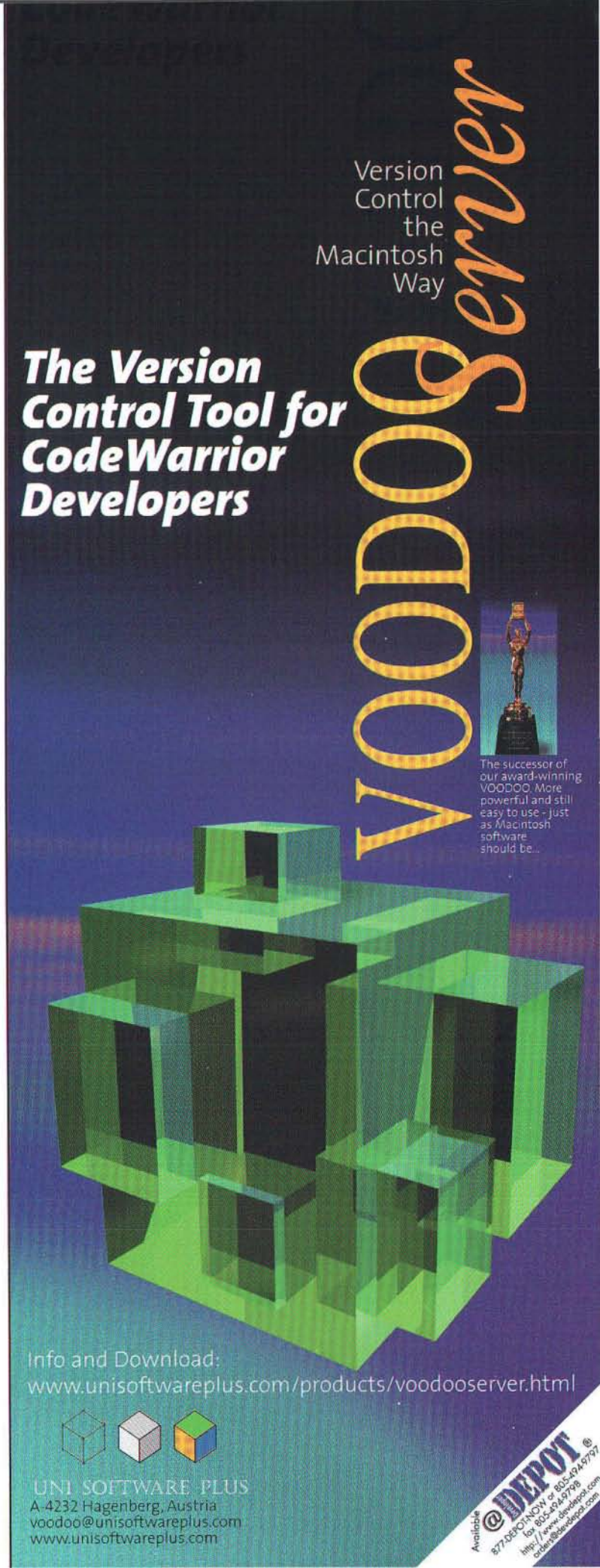


Figure 3. *The SpeechVoice dialog box.*

In the SpeechVoice dialog box you click on one of two radio buttons to select a voice to use, then click the Play Speech button to have the program speak the string *"Is this the voice you're looking for?"* using the selected voice. If the user's machine doesn't hold a voice that has the specified characteristics, the program instead uses the system default voice. You can repeat the process of choosing a voice and speaking the phrase as often as you wish. When finished, click the Quit button to end the program.

Creating the SpeechVoice Resources

Start your resource development by creating a new folder named SpeechVoice in your main CodeWarrior folder. Start ResEdit and create a new resource file named SpeechVoice.rsrc. Specify that the SpeechVoice folder act as the resource file's destination. This resource file requires four resources, two of which you're very familiar with: the one ALRT and one of the two DITLs. ALRT 128 and the corresponding DITL 128 together define the program's error-handling alert. If the SpeechVoice program doesn't experience a serious error while executing, then this alert won't be seen by the user.

The remaining two resources are DLOG 129 and its corresponding DITL 129. Together these resources define the dialog box shown back in **Figure 3**. The size, placement, and type of dialog that the DLOG defines aren't too critical, though it makes sense to select the type that doesn't have a close box (since the dialog is to remain on-screen until the user quits). Figure 4 shows the DITL that defines the type and placement of the items in the dialog box. To view the item numbers (as displayed in **Figure 4**), check Show Item Numbers from ResEdit's DITL menu. Take note of the item numbers — they'll appear as constants in the source code.

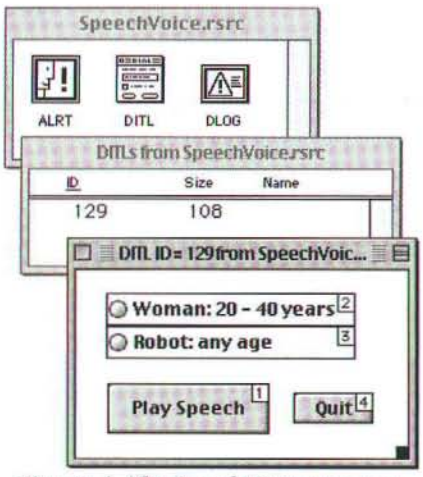


Figure 4. *The SpeechVoice resources.*

Creating the SpeechVoice Project

Create a new project by launching CodeWarrior and choosing New Project from the File menu. Use the MacOS:C_C++:MacOS Toolbox:MacOS Toolbox Multi-Target project stationary for the new project. Uncheck the Create Folder check box, then click the OK button. Name the project SpeechVoice.mcp and choose the existing SpeechVoice folder as the project's destination.

Add the SpeechVoice.rsrc resource file to the project, then remove the SillyBalls.rsrc file. You can remove the ANSI Libraries folder if you want as the project won't be making use of any ANSI C libraries.

If you plan on making a PowerPC version or fat version of the SpeechVoice program, add the SpeechLib library to the two PowerPC targets of your project. As mentioned in last month's *Getting Started*, you'll want to choose Add Files from the Project menu and then maneuver your way to this library. The most likely spot to find this library is in the Metrowerks CodeWarrior:MacOS Support:Libraries:MacOS Common folder. If it's not there, use Sherlock to search your hard drive. After adding the library to the project, CodeWarrior displays a dialog box asking you which targets to add the library to. Check the two PPC targets.

Now create a new source code window by choosing New from the File menu.. When you save the window, give it the name SpeechVoice.c. Choose Add Window from the Project menu to add the new empty file to the project. Now remove the SillyBalls.c placeholder file from the project window. You're all set to type in the source code.

If you want to save yourself a little typing, connect to the Internet and visit MacTech's ftp site at ftp://ftp.mactech.com/src/. There you'll find the SpeechVoice source code file available for downloading.

Walking Through the Source Code

SpeechVoice.c begins with the inclusion of the Speech.h file:

```
/****************** includes ******************/

#include <Speech.h>
```

After the #include comes a number of constants, most of which are resource-related. The constant kALRTResID holds the ID of the ALRT resource used to define the error-handling alert. kDLOGResID holds the ID of the DLOG resource used to define the program's dialog box. The constants kPlaySpeechButton, kSetSpeechWomanRadio, kSetSpeechRobotRadio, and kQuitButton each hold the item number of one of the items in the dialog box (compare these constants to the items in DITL 129 as shown back in **Figure 4**). The constant kNoMatchingVoiceErr is our own (arbitrary) value that we'll use in the event the program fails in its attempt to find an appropriate voice. The constants kControlOn and kControlOff are used in the turning on and off of the dialog box radio buttons.

```
/****************** constants ******************/

#define     kALRTResID              128
#define     kDLOGResID              129
#define     kPlaySpeechButton       1
#define     kSetSpeechWomanRadio    2
#define     kSetSpeechRobotRadio    3
#define     kQuitButton             4
#define     kNoMatchingVoiceErr     -999
```

```
#define     kControlOn              1
#define     kControlOff             0
```

Next come the program's function prototypes.

```
/****************** functions ******************/

void            ToolBoxInit( void );
void            OpenSpeechDialog( void );
SpeechChannel OpenOneSpeechChannel( VoiceSpec );
OSErr           GetVoiceSpecBasedOnAgeGender( VoiceSpec *,
                                        short, short, short
);
void            DoError( Str255 errorString );
```

The main() function of SpeechVoice begins with the declaration of three variables, all of which are used in the determination of whether speech generation is possible on the user's Mac.

```
/****************** main ******************/

void   main( void )
{
  OSErr  err;
  long   response;
  long   mask;
```

After the Toolbox is initialized the speech-related tests (as described in January's *Getting Started*) are made. After that main ends with a call to the application-defined function OpenSpeechDialog().

```
  ToolBoxInit();

  err = Gestalt( gestaltSpeechAttr, &response );
  if ( err != noErr )
    DoError( "\pError calling Gestalt" );

  mask = 1 << gestaltSpeechMgrPresent;
  if ( response & mask == 0 )
    DoError( "\pSpeech Manager not present " );

  OpenSpeechDialog();
}
```

ToolBoxInit() remains the same as previous versions.

```
/****************** ToolBoxInit ******************/

void   ToolBoxInit( void )
{
  InitGraf( &qd.thePort );
  InitFonts();
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs( nil );
  InitCursor();
}
```

OpenSpeechDialog() is responsible for opening, displaying, and monitoring the program's dialog box. It's also responsible for generating speech. The function begins with a host of variable declarations, each to be discussed at the time it's used in the function:

```
  DialogPtr         dialog;
  short             oldRadio;
  short             type;
  Handle            handle;
  Rect              rect;
  short             item;
  Boolean           done = false;
  OSErr             err;
  SpeechChannel   channel;
  Str255      str = "\pIs this the voice you're looking for?";
  short             ageLow;
  short             ageHigh;
  short             gender;
  VoiceSpec         defaultVoiceSpec;
  VoiceSpec         theVoiceSpec;
  VoiceDescriptionvoiceDesc;
```

Creating a new dialog box, and saving a pointer to it in variable dialog, is the first order of business:

```
  dialog = GetNewDialog( kDLOGResID, nil, (WindowPtr)-1L );
```

Both radio buttons will initially appear unselected, so it's up to us to turn one on. GetDialogItem() returns (among other pieces of information) a handle to the item named in the second argument. This generic handle is then typecast to a ControlHandle and used in a call to SetControlValue() to turn the radio button on. To let the routine know which radio button is now on, the variable oldRadio is set to match the item number of the just turned on button.

```
  GetDialogItem( dialog, kSetSpeechWomanRadio, &type,
                 &handle, &rect );
  SetControlValue( ( ControlHandle )handle, kControlOn );
  oldRadio = kSetSpeechWomanRadio;
```

Now we'll call ShowWindow() to handle the case of a DLOG resource that specified that the dialog box be initially visible. A call to SetPort() ensures that the newly opened dialog box is the window that receives updating.

```
  ShowWindow( dialog );
  SetPort( dialog );
```

Before jumping into the loop that will watch for, and handle, the user's actions, we need to take one more preliminary step. As shown earlier in this article, the GetVoiceDescription() function is usually called with a VoiceSpec as the first argument. If a value of nil is passed instead, then a VoiceDescription for the system default voice is returned. We'll do that here in order to get, and save, the default voice. If our later attempts to find a particular voice fail, we'll use the system voice.

```
  err = GetVoiceDescription( nil, &voiceDesc,
                             sizeof( voiceDesc ) );

  defaultVoiceSpec = voiceDesc.voice;
```

We now begin the while loop that executes until the user clicks the dialog box Quit button. When the user clicks on any one of the four items in the dialog box, ModalDialog() returns the item number of the clicked-on item. We use that returned value in a switch statement:

```
  while ( done == false )
  {
    ModalDialog( nil, &item );

    switch ( item )
    {
```

If the user clicks on the radio button labeled Woman: 20 - 40 years, then the following code executes:

```
      case kSetSpeechWomanRadio:
        GetDialogItem( dialog, kSetSpeechWomanRadio, &type,
                       &handle, &rect );
```

```
        SetControlValue( (ControlHandle)handle, kControlOn
);
        GetDialogItem( dialog, oldRadio, &type, &handle,
                       &rect );
        SetControlValue( (ControlHandle)handle, kControlOff);
        oldRadio = kSetSpeechWomanRadio;
        gender = kFemale;
        ageLow = 20;
        ageHigh = 40;
        break;
```

The above code turns the previously off button on, then turns the previously on button off (got that?). The newly turned on button is now considered the old button — which's information necessary for the next time a radio button is clicked. Three local variables, gender, ageLow, and ageHigh, are then set to values appropriate to the selected option. We'll use these values when the user eventually clicks the Play Speech button.

If the user instead clicks on the radio button labeled Robot: any age, then the following code executes. Recall that the Apple-defined constant kNeuter is used to specify a robotic voice. Since we've set up this option such that there's no age restrictions on the voice, we set the lowest acceptable age to 0 and the highest acceptable age to an arbitrarily large value (keeping in mind that a variable of type short can hold a value a little larger than 32,000).

```
case kSetSpeechRobotRadio:
  GetDialogItem( dialog, kSetSpeechRobotRadio, &type,
                 &handle, &rect );
  SetControlValue( (ControlHandle)handle, kControlOn );
  GetDialogItem( dialog, oldRadio, &type, &handle,
                 &rect );
  SetControlValue( (ControlHandle)handle, kControlOff);
  oldRadio = kSetSpeechRobotRadio;
  gender = kNeuter;
  ageLow = 0;
  ageHigh = 30000;
  break;
```

Clicking either of the radio buttons doesn't cause a search for the desired voice to take place. Instead, we wait until the user clicks the Play Speech button. When the user takes that step, we call our application-defined function GetVoiceSpecBasedOnAgeGender(). When we pass this function the address of a VoiceSpec, a range of ages, and a gender, the function searches the system for a voice with matching characteristics and fills in the VoiceSpec variable with the voice specification for that voice (more on this function ahead):

```
case kPlaySpeechButton:
  err = GetVoiceSpecBasedOnAgeGender( &theVoiceSpec,
                                      ageLow, ageHigh, gender
);
```

GetVoiceSpecBasedOnAgeGender() is written such that a failed attempt to find a voice results in the returning of an error value of -999, or kNoMatchingVoiceErr. In such a situation we set our local theVoiceSpec variable to the previously saved default voice. For any other error value we instead call our own error-handling routine that displays a short message and terminates the program.

```
  if ( err == kNoMatchingVoiceErr )
    theVoiceSpec = defaultVoiceSpec;
  else if ( err != noErr )
    DoError( "\pError finding voice" );
```

Now it's time to open a new speech channel. In last month's *Getting Started* we wrote OpenOneSpeechChannel() to handle that task. Here we call a slightly modified version of that routine. In this new version we pass a VoiceSpec along for use in opening the channel. This is the VoiceSpec returned by GetVoiceSpecBasedOnAgeGender() (or the VoiceSpec of the default voice in the event that a matching voice wasn't found).

```
  channel = OpenOneSpeechChannel( theVoiceSpec );
  if ( channel == nil )
    DoError( "\pError opening a speech channel" );
```

With a new speech channel open (and with that channel associated with the desired voice), it's time to test things out by speaking a phrase. Variable str holds the text to speak. The SpeakText() routine speaks the text of that string (refer to last month's *Getting Started* for the details on SpeakText()).

```
  err = SpeakText( channel, (Ptr)(str + 1), str[0] );
  if ( err != noErr )
    DoError( "\pError attempting to speak a phrase" );

  while ( SpeechBusy() == true )
    ;

  err = DisposeSpeechChannel( channel );
  if ( err != noErr )
    DoError( "\pError disposing speech channel" );
  break;
```

When the user is finished a click of the Quit button ends the dialog box loop, and ends the program.

```
    case kQuitButton:
      done = true;
      break;
  }
 }
 DisposeDialog( dialog );
}
```

OpenSpeechDialog() made calls to two application-defined routines: GetVoiceSpecBasedOnAgeGender() and OpenOneSpeechChannel(). Here's that first routine:

```
OSErr GetVoiceSpecBasedOnAgeGender(
                                VoiceSpec *theVoiceSpec,
                                short      ageLow,
                                short      ageHigh,
                                short      gender )
{
  OSErr             err;
  short             numVoices;
  short             i;
  VoiceDescriptionvoiceDesc;

  err = CountVoices( &numVoices );
  if ( err != noErr )
    return ( err );

  for ( i = 1; i <= numVoices; i++ )
  {
    err = GetIndVoice( i, theVoiceSpec );
    if ( err != noErr )
      return ( err );

    err = GetVoiceDescription( theVoiceSpec, &voiceDesc,
                  sizeof( voiceDesc ) );
    if ( err != noErr )
      return ( err );

    if ( (voiceDesc.age >= ageLow) &&
                                (voiceDesc.age <= ageHigh)
)
      if ( voiceDesc.gender == gender )
        return ( noErr );
  }
  return ( kNoMatchingVoiceErr );
}
```

We can get by with very little descriptive information about this routine because its code has already been described in this article. Recall that CountVoices() returns the number of voices in the user's system, and that number can then be used as a loop index. Each pass through the loop calls GetIndVoice() to obtain a voice specification for one voice, and then calls GetVoiceDescription() to get the voice description of that one voice. From the voice description it can be determined if a suitable match has been made. Here we look to see if the voice's age falls into the range of ageLow and ageHigh, and whether the voice is of the proper gender. If a match is made, return ends the loop. At this point theVoiceSpec holds the voice specification for a matching voice.

The second application-defined function called by OpenSpeechDialog() is OpenOneSpeechChannel(). As you saw in last month's article, this routine calls NewSpeechChannel() to create a new speech channel. Last month we passed nil as the first argument, telling NewSpeechChannel() to associate the system default voice with the channel. Here we pass in a voice specification and use that as the first argument, telling NewSpeechChannel() to associate this voice with the new channel.

```
/************** OpenOneSpeechChannel **************/

SpeechChannel OpenOneSpeechChannel( VoiceSpec theVoiceSpec )
{
  SpeechChannel channel;
  OSErr     err;

  err = NewSpeechChannel( &theVoiceSpec, &channel );
  if ( err != noErr )
  {
   err = DisposeSpeechChannel( channel );
   channel = nil;
  }

  return ( channel );
}
```

DoError() is unchanged from prior versions. A call to this function results in the posting of an alert that holds an error message. After the alert is dismissed the program ends.

```
/****************** DoError ******************/

void   DoError( Str255 errorString )
{
  ParamText( errorString, "\p", "\p", "\p" );

  StopAlert( kALRTResID, nil );

  ExitToShell();
}
```

Running SpeechVoice

Run SpeechVoice by choosing Run from CodeWarrior's Project menu. After the code is compiled, CodeWarrior launches the SpeechVoice program. After you're satisfied that the program does in fact select and use an appropriate voice, click the Quit button to quit.

Till Next Month...

Two months ago you saw how a Mac program can speak the text in a single string. Last month you learned how your program can use a buffer and a speech channel to speak larger amounts of text. And finally, this month you read up on how to select a voice and then use that voice in generating speech. You can learn still more about speech and voices by reading the *Sound* volume of *Inside Macintosh*. You can also learn more about voices by experimenting with the SpeechVoice code. For starters, try adding a third radio button that specifies different voice characteristics. By the time you have speech fully integrated into your own program, you'll be ready to read up on a new topic in next month's *Getting Started* article... MT

C++ Developer's Kit®

Advanced Tools for ANSI C++ Development

Professional includes
- 3 CDs
- 1 manual: "C++ Developer's Guide"
- 1 manual: "C++ Developer's Tools"
- 1 UML/C++ Quick Reference
- More of 250.000 C++ code lines
- UML standards
- Complete UML seminar on CD
- Industry standard STL library
- Major C++ Internet addresses
- OOSE, OOP features

FREE shipping with DHL (Europe CEE, USA and Canada only)

"The C++ Developer's Kit is a tutorial and reference collection showing programmers how to create better ANSI C++ code. It includes libraries of useful programming snippets and classes that you can use it your own projects to create more efficient, powerful, and reliable applications for any platform!"

Win Mac · ISO ANSI C++ · developer's C++news

TECHNOSOFT www.technosoftweb.com orders@technosoftweb.com

By Bill von Hagen

Put Your Network Under a Microscope With EtherPeek

Better living through network analysis software

Networking is arguably the most important innovation in modern computing. Networks are the backbones of most corporate and academic computing environments, plus access to the Internet is the most common reason for home computer purchases today. Unfortunately, diagnosing and resolving network problems poses a special set of headaches for system and network administrators. Bad network interface cards (NICs), bad connections, bad hubs, gateways, routers, or simply bad software can cause network problems.

It wasn't that long ago that the first line of defense for network problems was an armada of scopes, line testers, and other medieval devices to help track down the source of a problem. If the source of the problem was a computer system, system administrators could begin verifying the system's hardware, configuration, network software, and then make coffee and exchange hex dumps of network packets all night long. The next advance in network problem analysis was dedicated portable hardware that you could attach to your network at any suspected trouble spot. Often laptops running specialized software for capturing and analyzing network traffic were used. This is still a practical, sometimes necessary, option for organizations with extremely specialized needs (and unlimited budgets).

A more practical option is to buy EtherPeek, put if on a dedicated Mac, and dare problems to occur on your network segment. EtherPeek is a great example of just how much software can do to provide a comprehensive, understandable view of exactly what is happening on a network or network segment. EtherPeek makes it easy to identify and track down network problems before they bring your network to its knees. EtherPeek can also help you improve network performance by identifying hotspots or unnecessary network communication. It can identify abnormal traffic patterns and performance bottlenecks, and help you limit specific types of traffic to certain subnets by selecting systems to group on their own LAN segments. EtherPeek can also help you identify and eliminate unnecessary network communication, such as systems broadcasting or exporting network protocols that you don't use, increasing the bandwidth for the network communication that you actually want to take place on your network.

EtherPeek should only be installed on secure systems or servers to which access is restricted. It obtains the information it uses to analyze your network through what is called "packet sniffing." It grabs a copy of every network information unit (packet) on your network as it goes past, regardless of the system or network address they're intended for. (Ethernet interfaces normally only accept packets that are intended for their hardware address.) Packet sniffing is the most common way for hackers (or whatever term you prefer) to obtain logins, passwords, and other supposedly secure information by eavesdropping on network communication. General Network Diagnostics

EtherPeek provides a number of floating diagnostic windows that enable you to monitor the general performance and status of a network while you use its integrated capture and analysis commands to examine selected portions of those communications. When you first start EtherPeek, it displays a

Bill von Hagen is a writer, computer system administrator, and the author of "SGML for Dummies." You can contact him at wvh@gethip.com.

dialog in which you select the network interface that you want to monitor. After you select the appropriate network interface, you should also use the Capture menu's Network Speed command to verify that EtherPeek has detected the correct speed of your network (10 MB, 100 MB, or a custom OTHER setting). Correct this setting if necessary.

EtherPeek then displays a Network Statistics window that provides summary information about the amount of traffic on your network. It uses a speedometer/tachometer display to show network utilization and traffic at the current moment.

Once you're sure that you have EtherPeek listening on the correct network interface, the fun begins. First, obtain a summary of the network protocols that are in use on your network. Select the Statistics menu's Protocols command that displays a Protocol Analysis window. Protocol Analysis provides a constantly updated summary of the protocols in use on your network, grouping families of protocols together under their parent protocols. The Protocol Statistics window samples the packets on the selected network interface at a configurable rate ranging from every second to every ten minutes, and displays a running histogram of the percentage of different protocols and packet types seen on your network.

Figure 1 shows the Protocol Statistics window on a simple network where all of the current traffic consists of TCP/IP protocols such as, secure HTTP, and TELNET. This is the sort of display that you would see on a network composed primarily of UNIX systems, where none of the standard Apple or Microsoft protocols are being used.

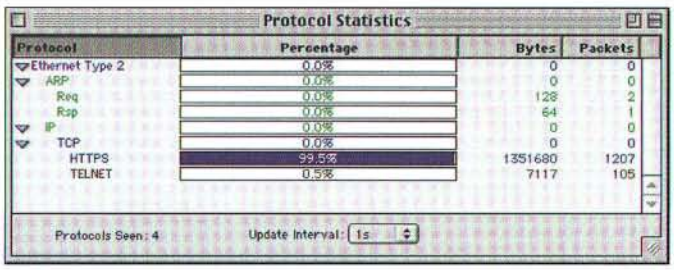


Figure 1. *The Protocol Statistics Window.*

To watch a specific family of protocols, click on the arrowhead beside other family names to hide the protocols in those families, which helps you focus on those that you are specifically interested. **Figure 2** shows a Protocol Statistics window after bringing up a PC with AppleTalk support on the same network, with all of the Ethernet Type 2 protocols collapsed. This shows Apple's AppleTalk protocol, the standard PC networking protocols (IPX, IPX-LSAP, NetBEUI/NetBios), and the Datagrams associated with PC Server Message Block (SMB) file and device sharing.

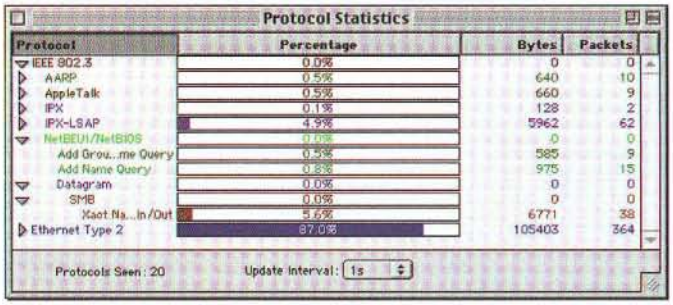


Figure 2. *Protocol Statistics Window With Additional Protocols.*

All of EtherPeek's statistics windows can be sorted by any field, just click on the heading for that field; this is a very nice feature. For example, you could click the Bytes header in the Protocol Statistics window to sort its contents by the number of bytes being sent within each protocol family. This makes it easy to see the heaviest used protocols on your network.

Protocol statistics can be quite useful to identify the types of traffic going out over your network and the amount of traffic using each protocol, but don't provide detailed information about the systems between which that traffic is taking place. The Statistics menu's Nodes command displays that information. Each host that is communicating on your network is identified by its MAC address, which is the unique hardware address of an Ethernet card. To make this information useful, each MAC address is followed by each protocol family address that communicates using that hardware address. **Figure 3** shows a sample Node Statistics window in which you can see that the MAC address 00:00:94:B5:72:A5 is receiving IP packets as the address 192.168.6.85, while the MAC address 00:00:C0:8C:CE:92 is handling both the AppleTalk address AT-65339.7 and the IP address 192.168.6.95.

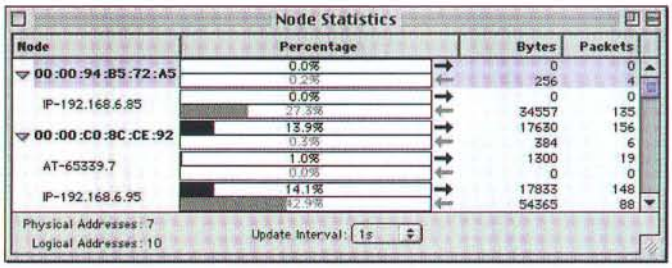


Figure 3. *The Node Statistics Window.*

The Node Statistics window makes it easy to verify which hosts are communicating on your network, which protocols they're using, and the amount of traffic to and from each node. This can be very useful in identifying whether heavy network users are using the recommended protocols. For example, many UNIX sites use software or special hardware that enables them to use AppleTalk over IP to spool print jobs from print servers to Apple LaserWriters on their networks. Any Macintosh on the same network can print directly to a LaserWriter by simply selecting it from the Chooser, which would disrupt or suspend the UNIX print spooling and would not be logged by the UNIX spooling software. On primarily UNIX networks, you could easily spot this sort of disruption by watching for any hosts that are not print servers but which are sending lots of AppleTalk traffic.

Identifying certain types of traffic can be useful, but the ability to identify the systems between which that traffic is flowing is usually more important in identifying performance problems. Select the Statistics menu's Conversation command to display a Conversation Statistics window that provides a summary of all communications between different network addresses. As shown in **Figure 4**, the Conversation Statistics window displays the source and destination nodes for each network conversation, the number of packets exchanged by those nodes, and the total number of bytes in those conversations. The Conversation Statistics window is updated every 5 seconds, by default, but you can change the update rate to any value between 5 seconds and one hour.

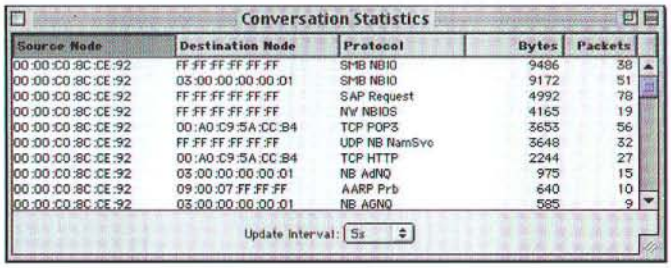


Figure 4. *The Conversation Statistics Window.*

Capturing and Analyzing Packets

EtherPeek's various statistics windows give you a great high-level picture of your network, but diagnosing actual network problems requires more information than summaries and reports can provide. EtherPeek enables you to capture a stream of packets and examine them in detail, so it is easy for you to actually see problems .

To capture packets, Select the File menu's New option to display the Capture Buffer Options dialog. The Capture Buffer Options dialog lets you define various options for the buffer in which EtherPeek temporarily stores any captured packets . Next click the Continuous Capture option and click OK. Once a packet capture window displays, you click the Start Capture button to actually start capturing packets. **Figure 5** shows a sample capture window containing traffic between hosts on a local network and actual Internet hosts.

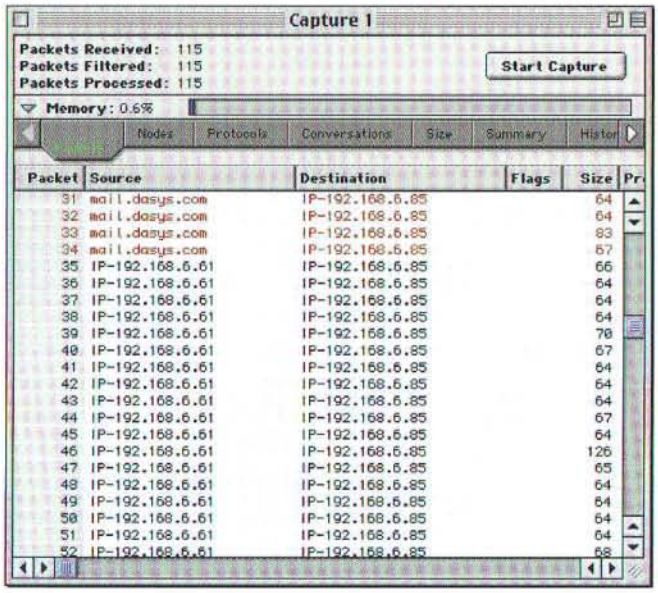


Figure 5. *A Packet Capture Window.*

Whenever possible, the packet capture window automatically uses DNS to resolve IP addresses into actual host names. **Figure 5** shows both IP addresses and host names because some of the systems on the test network use private IP addresses that can't be resolved through a global DNS server.

At any time during or after completing a packet capture session, you can examine the contents of a captured packet in more detail. Just double-click the packet you want to examine. This displays a window that provides detailed information about that packet, as shown in **Figure 6**.

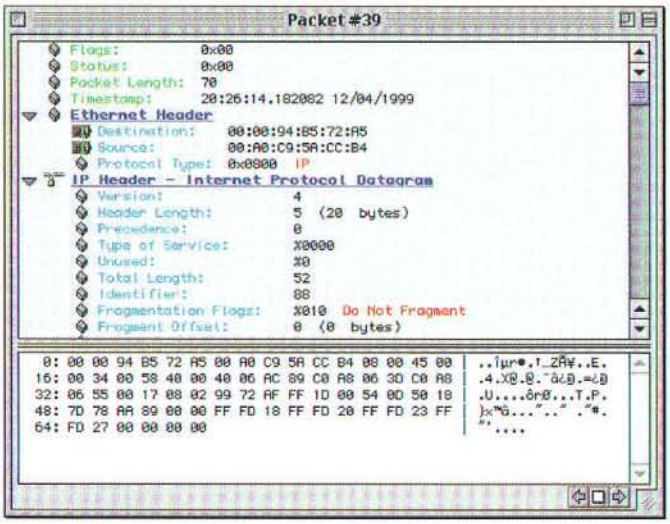


Figure 6. *Captured Packet Details.*

The detail window shows the contents of the selected packet, its header, and the actual contents of the packet in hexadecimal. You can click on any of the details shown in the top portion of a packet detail window to highlight the associated raw packet data.

Once you've captured the information you need, click the Stop Capture button to terminate the packet capture session. At this point, you can begin debugging or save the contents of the capture window for future analysis using the File menu's Save command.

Using Packet Filters

Capturing a constant stream of packets is a great diagnostic if you're examining general networking issues, but may often provide more information that you need. To limit captured packets, EtherPeek lets you create filters that identify the packets that you actually want to see. EtherPeek supports two classes of packet filters — simple filters let you capture packets based on specific parts of their header, and advanced filters let you create complex Boolean expressions based on header information or packet content. **Figure 7** shows EtherPeek's simple filter definition window.

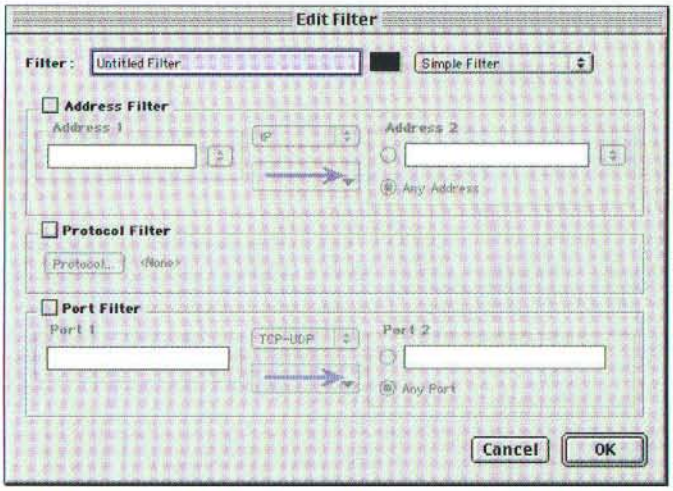


Figure 7. *The Edit Filter Dialog.*

EtherPeek's simple filters let you restrict the captured packets based on their source address, protocol family, or the port to which they are addressed. Port filtering is a great feature when trying to debug network problems related to a service such as FTP, Telnet, or Sendmail that uses a specific port.

EtherPeek's advanced filters let you define filters using Boolean combinations of the source address, protocol family, target port, packet value, packet contents, packet length, or different packet errors. Packet content filters are especially useful when watching for network attacks. You can limit the captured packets to those that contain specific string values, such as the password packets used by the TELNET and FTP protocols.

Automating Your Network Debugging

Although EtherPeek's filters make it easy to capture selected types of traffic in separate capture windows, constantly scanning a large number of open capture windows can be confusing. A better approach to look for specific events is to have EtherPeek automatically perform some action whenever the event occurs, such as beginning

a packet capture session or logging a message. EtherPeek provides three general ways of programming to automatically react to specific network events: Triggers, plug-ins, and notifications.

Triggers let you start or stop capturing packets at specific times or when specific network events occur, based on user defined filters. Time triggers make it easy for you to automatically snapshot network traffic associated with regularly scheduled events such as network backups or automatic system reboots. These are especially handy when you need to capture traffic snapshots for tasks scheduled at off-peak times, such as 4 AM, when you might prefer to be home sleeping. Filter triggers begin or end a packet capture session whenever the conditions defined in a specific filter are met.

Plug-ins are EtherPeek's most powerful method of reacting to specific events. Plug-ins are external modules that perform detailed packet analysis and constantly monitor all seen packets . EtherPeek comes with a number of plug-ins that perform tasks such as, looking for duplicate IP addresses, watching for specific types of network attacks, and analyzing FTP, NetWare, and SMTP (email) traffic. Once enabled, plug-ins are globally active; that means the traffic being captured by a filter trigger can be simultaneously analyzed by any number of plug-ins.

Both triggers and plug-ins have associated severity levels, which make it easy to integrate them with EtherPeek's notifications. Notifications are actions that you associate with specific events in EtherPeek, and consist of any of five different actions. You can log messages to EtherPeek's log file, be paged (if you're using supported paging server software, which is currently PageNOW!), be sent email, use the Speech Manager to announce the event, or launch an AppleScript that you create to do anything you want. Notifications also have associated severity levels: informational, minor, major, and severe. makes It is easy to create notifications with your own names, select the action you want to associated with them, and associate them with a specific severity level in EtherPeek's Notifications dialog . After notifications have been defined, any trigger or plug-in with a specific security level automatically triggers any notifications associated with that severity level. By default, EtherPeek comes with one predefined notification associated with all four severity levels – logging messages to its log file.

Generating Reports

System and network administrators rarely have the luxury of sitting at a desk to just monitor EtherPeek statistics windows. Today's networked computing environments require office to office or wiring closet to wiring closet travel, software and hardware installation and , or simply debugging user problems. To help keep an eye on your network while you're off fire-fighting, or to get a high-level view of network performance for use at an upcoming meeting, you can set EtherPeek to automatically generate statistical reports in HTML format.

The Statistics menu's HTML Output command lets you configure the frequency with which these reports are generated, the directory where the output files are placed, and a directory of report generation templates. EtherPeek generates Protocol Statistics and Node Statistics reports, plus a summary report that provides general information about all of the traffic on your network. The Protocol and Nodes Statistics reports exactly match the current view in the windows on your screen. For example, if you've collapsed or sorted families within the Protocol Statistics window, the reports generated by EtherPeek are collapsed or sorted in the same way.

Documentation

EtherPeek is a rarity among modern software packages because it comes with a well written, printed manual. You should think twice before buying a diagnostic tool that only comes with PDF manuals or other online documentation. Online documents can be tricky to access when you're experiencing hardware problems. EtherPeek's documentation not only explains the buttons and how to use the dialogs, but also includes multiple examples of how to use EtherPeek to monitor your network and troubleshoot various types of problems. If you are new to network troubleshooting, need some suggestions or just want a sanity check of your own approaches to troubleshooting, this information is invaluable.

Diagnosing the Diagnostics

Diagnostic tools are a tough business because they have to provide useful information when things are going well, handle anomalous events with ease, and provide information about those anomalies in a useful form. The last thing you need when experiencing a network problem is to have problems with your diagnostic software. Luckily, when testing EtherPeek, I got a first-hand opportunity to submit a problem report and watch AG Group's customer support staff in action.

One of the systems on my test network dual-boots Windows 98 and the BeOS. The BeOS is somewhat picky about Ethernet card support, so I have two network cards in the machine, one for each operating system with the other disabled by that operating system. When the machine comes up, the hub to which these cards is connected erupts in a flood of blinking lights, indicating chatter between the two cards until the OS de jour shuts down one of them. A perfect test! I started EtherPeek on a Mac, opened a packet capture window, and then fired up the dual-boot system and let the chattering begin.

Within seconds, the Protocol Statistics window had reported over 4000 different detected protocols , a big change from the 30 or so seen in normal circumstances. Each of the bogus protocols had a name of the form ETHER-XX-XX, where X was a hex digit. Capturing all of these packets also brought the Mac to its knees. The only way I could successfully stop the capture and save the contents of the capture buffer was to turn on continuous logging to disk and shut down the BeOS/Win98 box a second or two after turning it on. Looking at the capture buffer, I could see that one of the cards was sending thousands of undersized and otherwise bogus packets.

AG Group's customer support was quick to tell me that the ETHER-XX-XX packets meant that the value in the type field for those packets was not a known protocol type. They also explained how to set up an Advanced Filter to ignore undefined packet types, and even offered me a copy of an incremental release of EtherPeek with some new features to improve performance in heavy traffic situations, such as mine. (Coming soon to the next version of EtherPeek!) Admittedly, I identified myself as someone who was reviewing their software and thus your mileage may vary. The speed with which they responded and the quality of their response meant that the support person I contacted actually understood both their software and networking in general.

Diagnostic tools can't fix hardware problems, but they can tell you when to punt a cheap network card. In a real business situation, my NIC could have caused real performance problems for other users. Final score: cheap NIC, 0, EtherPeek and AG Group's support staff, 10.

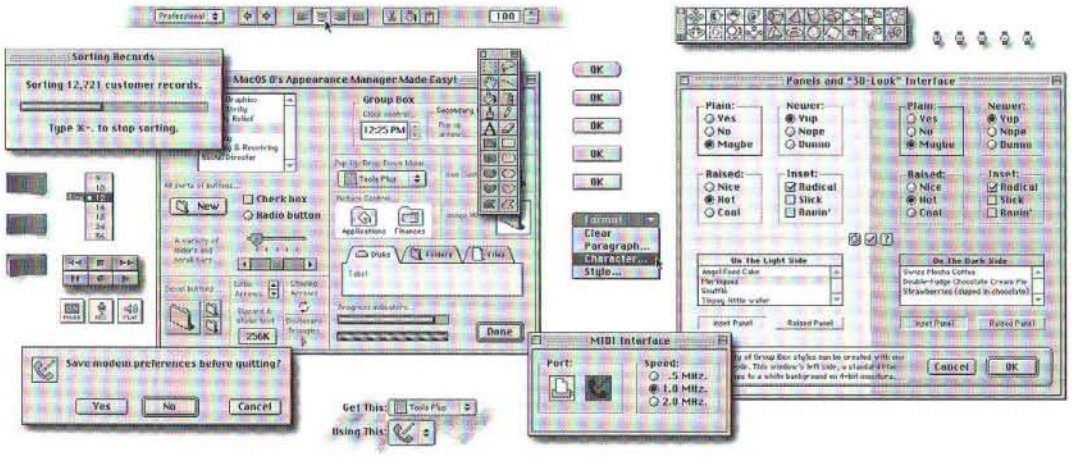
Conclusion

Network problems are hard to identify and often harder to track down to a specific host or network device. EtherPeek simplifies network troubleshooting by providing excellent packet capture and analysis capabilities within a powerful, well-designed graphical interface. If you manage a network, EtherPeek provides the capabilities and features you need. If you are in the network driver or network software business, EtherPeek should be a fundamental part of your arsenal of debugging and verification tools. Beyond simply diagnosing problems, EtherPeek is also an excellent tool for enhancing network performance by identifying local network traffic that you can isolate, and by helping you eliminate wasted traffic or questionable hardware.

EtherPeek can turn any Mac on your network into a powerful network debugging and monitoring station capable of extracting admiration and sysadmin envy from even the most hardened UNIX geek. If your organization doesn't already have Macs, EtherPeek might just be the best argument you could offer for buying one. If you don't have Macs and your management won't let you buy a superior machine, EtherPeek is also available for Windows systems. MT

Mac programming is complex.
The answer is easy.

Tools Plus
PROFESSIONAL
LIBRARIES + FRAMEWORK



Rapid Development in *your* language.

Tools Plus Pro quickly turns your great ideas into great applications using C/C++ & Pascal.

Everything works as soon as you create it. It's easy to learn, really easy to use, and a breeze to maintain.

"Tools Plus 4.0's highly optimized pieces of plain text code are better than any visual programming alternatives... If you work in C/C++ or Pascal, Tools Plus will help you produce better shareware and commercial applications."
Macworld

"All in all, it's and incredibly rich collection of tools... if you are interested in developing applications that have "quality" written all over them, then Tools Plus is for you."
MacTech

"...the routines are more compact and faster than anything you might write... Every element of Tools Plus is useful... a bargain compared with coding these routines yourself."
Macworld

Tools Plus Pro Supports:
- CodeWarrior, Symantec and THINK Pascal
- C, C++ and Pascal
- 680x0 and PowerPC
- Mac OS 8 and System 7

Includes megabytes of custom windows, controls, fonts and other valuable resources.

Still only
$249
(Academic license, just $99!)

Macworld 4½

Best Rapid-Development Tool

MW ★★★★★

Water's Edge Software
2441 Lakeshore Rd. W
Box 70022
Oakville, Ontario
Canada L6L 6M8

Orders & Inquiries:
Phone: (416) 219-5628
Fax: (905) 847-1638

Also available at Developer Depot:
Phone: (800) 622-3381

Email: wateredg@interlog.com
Free Evaluation Kit: http://www.interlog.com/~wateredg

By Tom Djajadiningrat

Control Freaks Alert!

A clean interface to Beehive Technologies' ADB I/O using object-oriented Lingo

Introduction

The ADB I/O by Beehive Technologies is a device which you can connect to a Macintosh to communicate with external electronics. Simply put, the ADB I/O provides digital inputs (to read switches), digital outputs (to switch things on or off) and analog inputs with 8-bit resolution (to read all kinds of sensors). For the Macintosh platform, it has opened up areas of application to mainstream users that were previously the sole territory of electronic hobbyists or users with high end data acquisition hardware. A few of those areas are home automation, robotics, kiosks, scientific experiments and product simulation.

Many applications can communicate with the ADB I/O. One particularly powerful combination is the ADB I/O and Macromedia Director. Even if you intend to control your ADB I/O through AppleEvents or C/C++, you may find it worthwhile to test your set-up with Director. Director is easy to program, especially when it comes to the graphical user interface.

There is of course a downside to Director's ease of use: the temptation of sloppy programming. Director does not require you to declare local variables or provide function prototypes. Add Director's cryptic error messages and your program can turn into a tangled mess which is difficult to debug. While this is of course undesirable for any program, with a program that communicates with external devices, for example motors, the consequences of messy programming may be more severe.

This article aims to help you in battling messy Director code. By using Director's object-oriented features you will get to build a friendly and clean interface to the ADB I/O. In addition, in this article consistent naming for variables is used, which may be of help to you in writing your own scripts.

Required Hardware and Software

Required hardware:

- A Macintosh with an ADB port or a Macintosh with a USB port and a Griffin Technology iMate USB to ADB adapter.
- A Beehive Technologies' ADB I/O.

Required software:

- ADB I/O XCMDs 1.0.1. This file comes with the ADB I/O. It is also downloadable from the Beehive Technologies website.
- Macromedia Director 5 or 6 (Director 7 is not compatible with the ADB I/O XCMDs 1.01 file. A Director 7 compatible Xtra for the ADB I/O is currently in beta and downloadable from the Beehive Technologies website). Though this article explains things for Director 6, the differences with Director 5 are minimal and the sample files which accompany this article are provided in both Director 5 and 6 versions.

What You Need to Know

You need to be familiar with Macromedia Director and its scripting language Lingo. You need not have worked with object-oriented Lingo. What you need to know about object-oriented

Tom spent a large part of his life trying to work out which letter groupings minimize confusion when spelling out his name on the phone. Now that he has found that Djaj-adi-nin-grat leads to significantly better results than Dja-ja-di-nin-grat, he can fully concentrate on his research on interaction design at Delft University. Since he does not own any cats, you won't find any pictures of them on www.io.tudelft.nl/research/vormtheorie/djajadiningrat.html.

Lingo will be explained. You should be familiar with the capabilities of the ADB I/O, though a summary of what you need to know is provided.

Contents

We will start with an introduction to object-oriented Lingo. We will then cover the ADB I/O's hardware and software details relevant to this article. If you are already comfortable with these subjects, you may wish to skip them. Once we have established this ground, we start with building a Director Lingo object which forms a friendly and clean interface to the ADB I/O. At the end of this article you will find a section on troubleshooting and a section with suggestions for enhancements.

Object-Oriented Lingo

Instead of starting with a theoretical discussion of the advantages of object-oriented Lingo, we will dive straight in and put together a simple object-oriented Director movie. I think you will find that the virtues of object-oriented Lingo are much easier to understand once you have seen a concrete example. Our movie consists of three scripts: a parent script, a movie script and a frame script. First we will write the scripts, then we will test and discuss the resulting movie. Don't worry too much if you don't completely 'get it' when we write the scripts. All will be explained.

The parent script

A Director object is described in a parent script. Create a new movie first, then create a parent script by choosing Script from the Window menu (**Figure 1**). The script window opens. Name the script by typing testParentScript in the field at the top of the scripting window (**Figure 2**). To make sure that the script is a parent script, click on the info button (**Figure 3**) and choose parent from the type pop-up menu (**Figure 4**). Click OK.

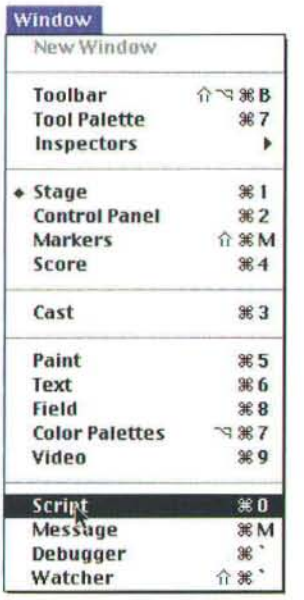
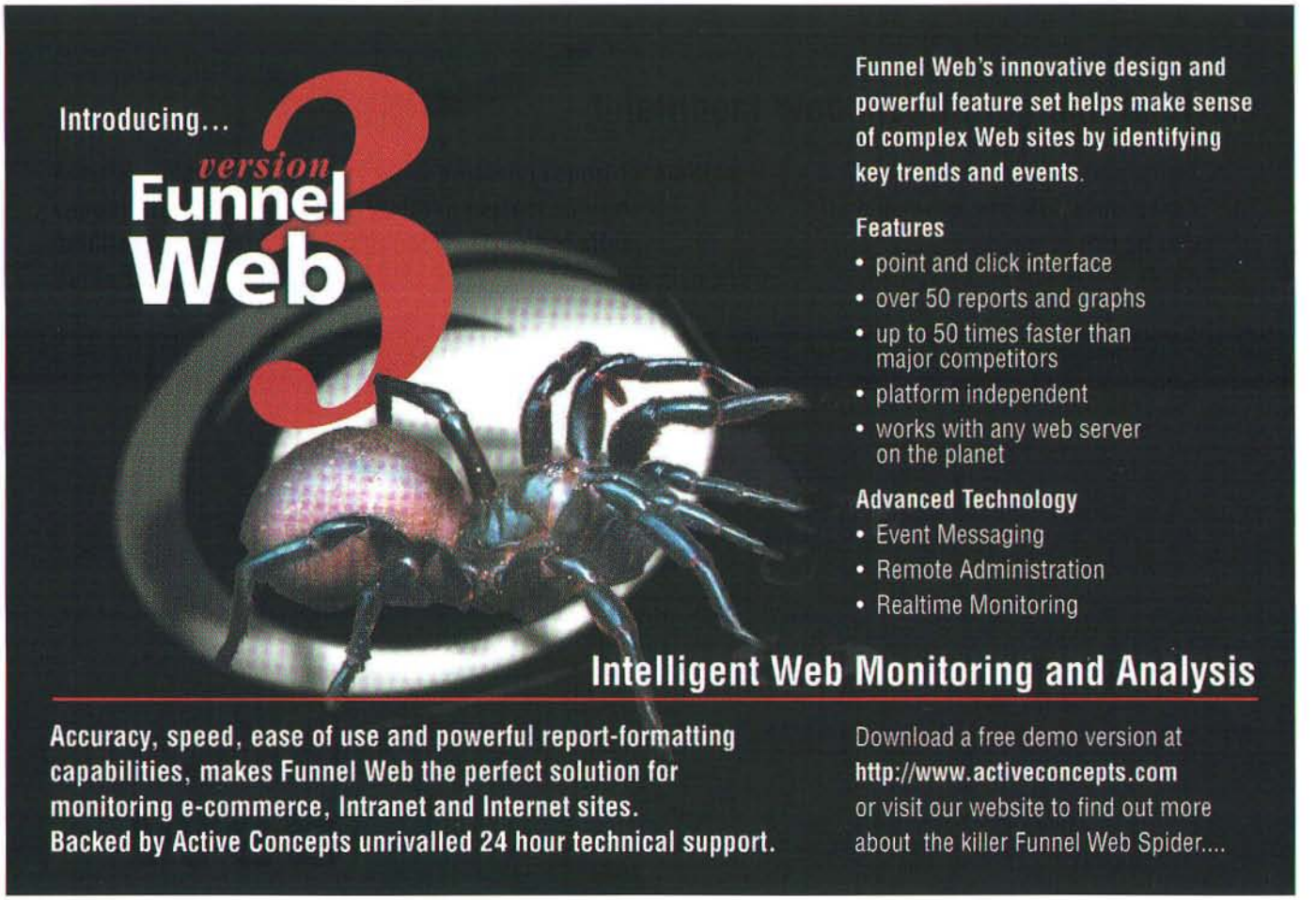


Figure 1. *Open the script window*



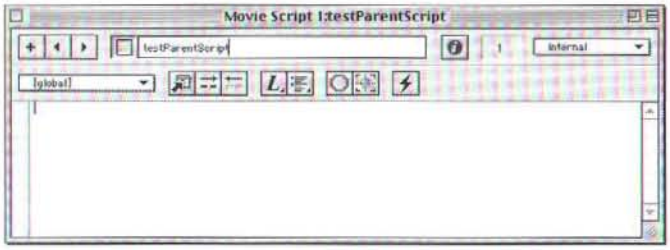


Figure 2. *Name the script*

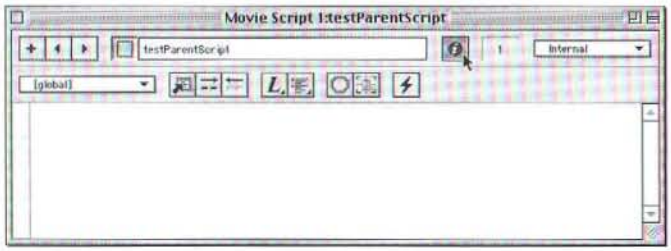


Figure 3. *Press the info button.*

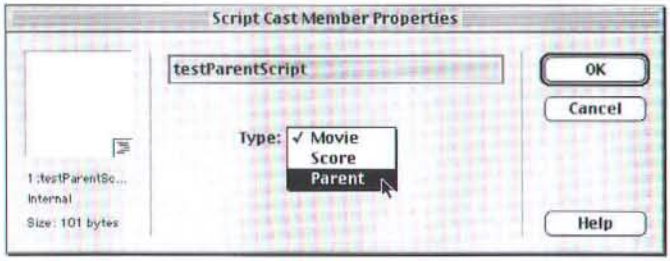


Figure 4. *Make the script a parent script.*

Now on to the parent script itself. It consists of three parts: a birthing handler, message handlers and properties. We will discuss each part in turn.

1. The birthing handler

This is where a child object is created. It typically has the form:

```
on new me
  return me
end
```

The birthing handler returns a pointer to a newly created child object. We will add one line of code to our new handler to put a message in the message box telling us that the birthing handler was indeed called.

```
on new me
  - print some feedback in the message window
  put "the birthing handler of testParentScript was called"

  - return a pointer to the new child object
  return me
end
```

2. Message handler

In a message handler you can specify a behaviour of an object. A message handler has the form:

```
on messageHandler me
end
```

We will create an message handler called hello which beeps and puts a message in the message box.

```
on hello me
  - beep once
  beep

  - print some feedback in the message window
  put "Hello World!"
end
```

3. Properties

A property is a variable that belongs to an object. It is declared at the top of an object's parent script, preceded by the Lingo keyword property, like this:

```
property pProperty
```

Within the parent script you can set or get a property by just its name:

```
set pProperty = 1
set lVariable = pProperty
```

When we test the Director movie, you will see how to get at a property from outside of the parent script. Notice how, to maintain an overview, we start a property with the letter p and a local variable with the letter l. For a full description of our naming convention for variables and parameters, please refer to Table 1.

Prefix	Suffix	Meaning	Example
i		input parameter	iChannel
l		local variable	lResult
o		child object	oADB1
p		property variable	pUnit
	List	list variable	pChannelTypesAList

Table 1: Variable and parameter naming conventions

For now, we will create a single property called pProperty and two member functions. One to set the value of pProperty and one to get the value of pProperty.

```
property pProperty

on setProperty me, iValue
  set pProperty = iValue
end

on getProperty me
  return pProperty
end
```

The movie script

To create the movie script, bring up the script window if necessary and click on the + button to create a new script. Name the script movieScript by typing into the field at the top of the script window. Click on the info button. A dialog box appears. Make sure that movie is selected in the type popup-menu. Click OK.

In our movie script we will write two handlers. The first one, startMovie, is called when the movie starts, the second one, stopMovie, is called when the movie stops.

```
on startMovie
end

on stopMovie
end
```

In this example we will add some code to the startMovie handler only. First we print some feedback to the message window to show that the startMovie handler was called. We then create a global called oTestObject1. Finally, we assign to this global a child object of the testParentScript that we wrote previously by calling the birthing handler.

```
on startMovie

  -- Print some feedback in the message window
  put "startMovie was called."

  -- create a global oTestObject1
  global oTestObject1

  -- make oTestObject1 a child object of the parentScript testParentScript
  -- by calling the birthing handler.
  set oTestObject1 = new (script "testParentScript")

end
```

The frame script

We will now create a simple frame script to make our Director movie loop in frame 1. In the score window, double click on the score channel of frame 1 and add one line of code to the exitFrame handler.

```
on exitFrame
  go to the frame
end
```

Testing our movie

It is time to test our movie. If you get stuck at any point and think it is because you did not quite get the explanation above, you may wish to look at testOOMovie.dir. Run the movie and open the message box. Type

```
put oTestObject1
```

The message box should display

```
-- <offspring "testParentScript" 2 876a72a>
```

indicating that oTestObject1 is a child object based on our testParentScript. Don't worry if the alpha-numeric string that you get differs from the one shown above. As this is a memory address it is in fact highly unlikely that it would be identical.

Now try the hello message handler. The syntax for sending a message to a child object is <messageHandler> <childObject>.

```
hello oTestObject1
```

The Mac should beep and the message box should display

```
-- "Hello World!"
```

Now send the getProperty message to our child object. Since getProperty is a function (a handler which returns a value), you need to enclose its parameters in round brackets.

```
put getProperty (oTestObject1)
```

The message box shows that pProperty has the value void. No worries, pProperty has not been set yet. Now type:

```
setProperty oTestObject1, 1
```

Now try again:

```
put getProperty (oTestObject1)
```

And you will see that pProperty has changed its value from void to 1.

setProperty and getProperty are message handlers which we wrote ourselves. Another way to set and get properties of a child object outside of its parent script is:

```
set the <property> of <childObject> = <value>
set lVariable = the <property> of <childObject>
```

Notice how we need to add the Lingo 'the' keyword, now that we try to access a property outside of its parent script.

Let's try:

```
set the pProperty of oTestObject1 = 2
put the pProperty of oTestObject1
```

The message box shows that pProperty of oTestObject1 now has the value 2.

Play around with accessing properties through the message handlers setProperty and getProperty as well as with accessing the properties directly through the 'the' keyword. You will find that you can mix both methods.

What's the point of object-oriented Lingo?

Before we answer that question, try creating a second child object:

```
set oTestObject2 = new (script "testParentScript")
```

The message box responds with

```
-- "new handler of testParentScript was called"
```

Try entering

```
put oTestObject2
```

The message box responds with

```
-- <offspring "testParentScript" 2 876a6ee>
```

Notice how the address of oTestObject2 differs from the address of oTestObject1. The values of the properties are also individual to each child object. Try

```
setProperty oTestObject1, 7
setProperty oTestObject2, 666
```

Now type:

```
put getProperty (oTestObject1)
```

The message box shows that the property pProperty is now 7. And try:

```
put getProperty (oTestObject2)
```

This time the message box responds that the property pProperty equals 666. That is pretty kewl: both child objects have a property called pProperty but each can hold a different value.

This example illustrates the three characteristics of object-oriented Lingo that we use in this article. The first is the ability to create multiple child objects based on the same parent script, which are all capable of listening to certain message through the message handlers defined in the parent script. The second characteristic is the ability to hide code — in our case a simple put statement — within the birthing handler of a parent script. This code is automatically executed when a child object is created. The third characteristic is the ability to assign individual property values to child objects. While these characteristics may seem trivial, we can use them to great effect in improving the cleanliness of our code. By hiding set-up details in the birthing handler, these details are prevented from littering your main program. Through the use of properties, variables can be encapsulated within an object. This greatly reduces the need for global variables. Global variables are a major source of bugs in Director because each time you forget to declare a variable as a global, Director will assume you are creating a new, local variable.

You have now seen enough of object-oriented Lingo to be able to understand the ADB I/O parent script that we will build later on. However, there is a lot more to object-oriented Lingo than you have seen so far. To learn more, you can have a look at Director's on-line help under parent script or have a look at Small (1999).

If you are familiar with some other object-oriented language such as C++ or Java, you will probably have worked out the relationship between Lingo and general object-oriented terminology by now. A parent script is a combination of a source and a header file in which you define an object, a birthing handler is a constructor, a message handler is the equivalent of a member function, a property is a data member and a child object is an instance.

The ADB I/O Hardware

Units

You can have a maximum of four ADB I/O units daisy chained to your Mac. Each ADB I/O needs to be set to its own unique ID number.

Ports

The ADB I/O has two ports, called port A and port B. Each port has four channels. The channels of port A and B differ in their capabilities.

Port A

Each of the four channels of port A can be configured as either a digital input or a digital output. Analog input is not possible on port A. Each channel can have either a relay or an opto-isolator installed. The ADB I/O comes with relays pre-installed on all channels of port A, making them suitable for digital output. If you wish to use any of these channels for digital input, you will need to replace a channel's relay by an opto-isolator. Through the use of relays or opto-isolators, external electronics are electrically isolated from the ADB I/O.

Port B

port B allows not only digital input and digital output, but also analog input and an analog reference voltage. However, you cannot freely mix all four channel types. Only the following configurations are allowed.

analog in, analog in, analog in, analog in
analog in, analog in, analog in, reference
analog in, analog in, digital in/out, digital in/out
digital in/out, digital in/out, digital in/out, digital in/out

Each channel of port B is equipped with a jumper. A jumper can be set to pull up, pull down or open. For digital input, the jumper needs to be set to either pull-up or pull-down. Note that to switch a channel from digital to analog, you not only need to change the configuration of port B through software, but you also need to move the jumpers.

The ADB I/O Software

To use the ADB I/O with Director 5 or 6, XCMDs are used. These come in a file called ADBIO XCMDs 1.01. You need to make sure that this file resides in the same folder as your Director movie. We can have a look at the functionality offered by the ADBIO XCMDs 1.01 file by creating a simple Director movie (If you get stuck during the creation of this movie you can take a peek at testXCMDs.dir). Start a new movie, create a movie script and add startMovie and stopMovie handlers like this:

```
on startMovie
  openXLib "ADB I/O XCMDs 1.0.1"
end

on stopMovie
  closeXLib "ADB I/O XCMDs 1.0.1"
end
```

Add a frame script in the score channel of frame 1 to loop the movie. Run the movie and open the message box. Now type

```
showXLib
```

A slew of Xtras, XCMDs and XFCNs show up. We are only interested in the ones that concern the ADBIO:

```
- "ADB I/O XCMDs 1.0.1"
- XCMD: Flash      Id:0
- XCMD: Picture      Id:4
- XCMD: Palette      Id:5
- XCMD: ADBReInit      Id:6701
- XCMD: CopyRes      Id:1077
- XCMD: ConfigureADBIO    Id:1200
- XCMD: SetADBIO       Id:1203
- XFCN: GetADBIO       Id:1201
```

For now let's look at ConfigureADBIO, SetADBIO and GetADBIO. Here only a short description of these calls is given. For a full description please refer to the ADB I/O manual.

ConfigureADBIO

- ConfigureADBIO has the syntax:

```
ConfigureADBIO <Unit> ,<port>,¬
<Channel1Type>, <Channel2Type>,¬
<Channel3Type>, <Channel4Type>
```

- **What does it do?**
  With ConfigureADBIO you specify what function the four channels of a particular port of a particular ADB I/O unit should fulfill.

- **Parameters**
  The first parameter, Unit, is used to address a particular ADB I/O unit and can range from 1 to 4. The second parameter, port, is used to address either port A or port B of an ADB I/O. The remaining parameters specify what channel type you wish to assign to each channel of the port. Keep in mind that the capabilities of the ports differ. If you call ConfigureADBIO with unallowed channel types it will return an error message.

- **Return values**
  ConfigureADBIO returns either 0 (success) or an error string (failure).

SetADBIO

- SetADBIO has the syntax:

```
SetADBIO <Unit>, <Port>, <Channel>¬
<Number> [,<'high'|'low'>] [,<Time>]
```

- **What does it do?**
  You can think of this call as three separate calls. All three versions apply to digital outputs only. The first version sets a channel of a port of a unit to high or low:

```
SetADBIO <Unit>, <Port>, <Channel>, <'high'|'low'>
```

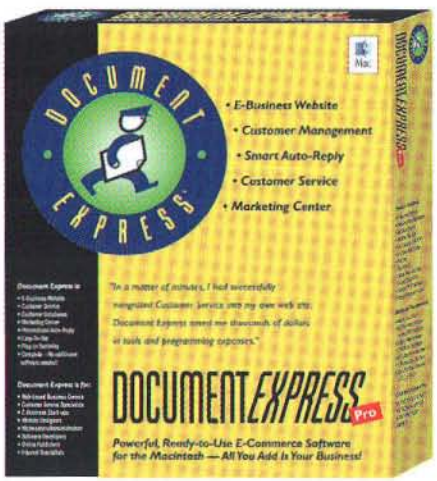
The second sets a channel of a port of a unit to high or low for a certain duration, after which it returns to the opposite state:

```
SetADBIO <Unit>, <Port>, <Channel>, <'high"|'low'>, <Time>
```


The most powerful Email and Webform management tool available!

Document Express is:
- E-Business Website
- Customer Service
- Customer Databases
- Marketing Center
- Personalized Auto-Reply
- Plug-in Extensible

Docment Express is for:
- Webmasters
- Site Administrators
- Software Developers
- Internet Specialists
- E-Business Startups
- Website Designers



Track file downloads, purchase requests, bug reports. Automatically generate serialization codes. Smart rules-based auto-reply and custom email routing. Prioritize customers and issues. Easy-to-use interface makes learning a snap.

Reduce support costs and increase customer satisfaction. Integrate Customer Service into your website immediately.

Personalize every automatic response. Increase customer satisfaction.
- Immediate acknowledgement.
- Automatically generate tracking numbers
- Never let a complaint or comment go unanswered
- Easily notify all your customers of special offers or updates

Account for each email message and every e-customer!
- Reporting of each Webform submission
- Increase quality control
- Automatically build customer base for online promotions
- Implement customer service and support right into your website
- Convert casual inquiries into sales opportunities
- Message customization to targeted customers
- Maximize customer retention

Attention, Metrowerks Code Warriors!!
Use the Document Express Module Development Kit with Code Warrior to create custom solutions just right for any business. Our software is fully extensible and flexible.

mac today HOT PICK!
macworld expo • new york • 1999

Document Express
7911 Herschel Ave., Suite 409, La Jolla, CA 92037 • **858-551-6767**

Download a free demo version at:
http://www.documentexpress.com

Available @ DevDepot
877.DEPOTNOW or 805.494.9797
fax 805.494.9798
http://www.devdepot.com
orders@devdepot.com

The third sets a port of a unit in such a way that the values of its channels form the binary equivalent of a decimal number:

```
SetADBIO <Unit>, <Port>, <Number>
```

- **Parameters**
  You already know the meaning of the Unit and Port parameters. Channel specifies the number of a channel and should therefore be in the range 1-4. 'high' means relay actived on port A and +5V on port B, 'low' means relay non-active on port A and 0V on port B. Time specifies a duration from 0-25.5 s in 0.1s increments through a integer ranging from 0 to 255. Number ranges from 0 (all channels 'low') to 15 (all channels 'high').

- **Return values**
  SetADBIO returns either 0 (success) or an error string (failure).

GetADBIO

- GetADBIO has the syntax:

```
GetADBIO(<Unit>, <port>[,'Number'])
```

- **What does it do?**
  You can think of this call as two separate calls. The first version:

```
GetADBIO(<Unit>, <Port>)
```

  returns a channel record with the channel type and channel value for each channel of a port:

```
<Channel1Type>,<Channel1Value>
<Channel2Type>,<Channel2Value>
<Channel3Type>,<Channel3Value>
<Channel4Type>,<Channel4Value>
```

  The second version:

```
GetADBIO(<Unit>, <port>, <"Number">)
```

  returns the decimal equivalent of the binary pattern on the channels of a port.

- **Return values**
  GetADBIO returns four lines of channel type and channel value combinations (success) or a decimal number in the range 0-15 (success) or an error string (failure).

Trying it out

Make sure you have at least one ADB I/O unit hooked up to your Mac, run the Director movie you just created and open the message box (Note that according to the ADB I/O manual, your Macintosh should be turned off before you attach or disconnect an ADB I/O). Play about with the three main calls ConfigureADIO, SetADBIO and GetADBIO. For each of these calls, make sure you type the Director keyword put in front of it and that you enclose the call's parameters in brackets so that you get to see the return value in the message box. For example, type:

```
put configureADBIO (1, "A",¬
"digital out", "digital out",¬
"digital out", "digital out")
```

No error should occur and the XCMD should answer with 0. Now be evil and force some error messages, for example:

```
put ConfigureADBIO 1, "A",¬
"analog in", "analog in",¬
"analog in", "analog in"
```

The return value is an error message:

```
- "Error : Wrong combination of ChannelType parameters"
```

Clearly, the XCMD is not happy about analog in on port A because only digital in and out are allowed. Let's try something else:

```
put SetADBIO 1, "A", 5, "high"
```

Again the return value is an error message:

```
- "Error : Invalid Channel parameter"
```

This time the XCMD is not happy because there are only four channels on each port. Play about a bit more. Don't worry about the channel values in the channel record returned by GetADBIO. For the moment no external electronics need to be attached to the ADB I/O, so never mind the values of the channels and ports. Just try to get a general feel for the required parameters and for the return values of the three handlers. Notice how the error messages returned by the three handlers are actually pretty specific. We will make use of this when we build our Director object to control the ADB I/O.

A Director Parent Script for the ADB I/O

Now that we have covered the basics of both object-oriented Lingo and the ADB I/O, we can write a parent script to control the ADB I/O. The parent script is called ADB I/O and forms part of testADBI/OMovie.dir included with the accompanying code. First we discuss the handlers concerning initialization and configuration, then those for input and then those for output. Finally we try out the whole parent script.

Initialization and configuration

Listing 1 shows the parent script's properties, a birthing handler and a configurePort handler.

Listing 1

properties

```
property pDEBUGMessage, pUnit
property pChannelTypesAList, pChannelTypesBList
property pChannelValuesAList, pChannelValuesBList
```

new

```
This handler creates a child object and hides set-up details.
Example: set oADB1 = new (script "ADB I/O", 1,TRUE)
Possible return values: An object address (success) or 0 (failure).

on new me, iUnit, iDEBUGMessage

  — Requested error reporting.
  set pDEBUGMessage = iDEBUGMessage

  — We are working with an object per ADB I/O unit.
  set pUnit = iUnit

  — These two lists hold the channel types
  — for port A and port B.
```

```
  set pChannelTypesAList = []
  set pChannelTypesBList = []
  — These two lists hold the channel values
  — for port A and port B.
  set pChannelValuesAList = []
  set pChannelValuesBList = []

  — The default initialization of the hardware is:
  — "digital out" for all channels of port A and
  — "digital in" for all channels of port B.
  — Call ConfigureADBIO for both port A and port B
  — with this default initialization.
  — According to the ADB I/O manual this is good practice.
  — The settings can always be changed later.

  — Initialize port A
  — Unit pUnit, port A, all channels digital out
  set lResult = configurePort(me, "A",¬
                              "digital out", "digital out",¬
                              "digital out", "digital out")

  if (lResult <> 0) then
    if pDEBUGMessage = TRUE then put lResult
    return 0
  end if

  — Initialize port B
  — Unit pUnit, port B, all channels digital in
  set lResult = configurePort(me, "B",¬
                              "digital in", "digital in",¬
                              "digital in", "digital in")

  if (lResult <> 0) then
    if pDEBUGMessage = TRUE then put lResult
    return 0
  end if

return me
end
```

configurePort

```
This handler configures a port.
Example:
configurePort (oADB1,"A",  "digital out","digital out",¬
                           "digital out","digital out")
Possible return values:
Empty string "" (success) or an error string (failure).

on configurePort me, iPort, iChannel1Type, iChannel2Type,¬
                            iChannel3Type, iChannel4Type

  — ConfigureADBIO returns either an empty string
  — "" (success) or an error string (failure).
  put configureADBIO (pUnit, iPort,¬
                      iChannel1Type, iChannel2Type,¬
                      iChannel3Type, iChannel4Type)¬
                      into lResult

  if lResult = 0 then

    case iPort of
      "A": set pChannelTypesAList =¬
                  [ iChannel1Type, iChannel2Type,¬
                    iChannel3Type, iChannel4Type]¬
      "B": set pChannelTypesBList =¬
                  [ iChannel1Type, iChannel2Type,¬
                    iChannel3Type, iChannel4Type]
    end case

  else
    — An error occurred
    if (pDEBUGMessage=true) then put¬
    "ConfigureADBIO for port" && iPort && "of unit"¬
    && pUnit && "failed."
  end if

  return lResult
end
```

Starting with the birthing handler, we come across the parent script's properties straight away. pUnit holds an integer with the ID number of an ADB I/O device which we pass into the birthing handler through the parameter iUnit.

pDEBUGMessage is a boolean property which we can turn on to get a bit more debugging info in the message window. We can turn it on through the birthing handler's parameter iDEBUGMessage. The chosen approach to debug messages is to report nothing while there are no errors. Only if there is an error will a debug message be printed to the message box. This is because writing to the message box considerably slows down a Director movie. If there are no errors we want to avoid any slow downs as they may cause timing problems.

Next is a set of four lists. pChannelTypesAList is a list which holds the channel types for port A. Similarly, pChannelTypesBList holds the channel types for port B. pChannelValuesAList is a list with the channel values of port A. Finally, pChannelValuesBList is a list with the channel values of port B. All lists are initialized as empty lists.

We use the birthing handler to hide some of the port set-up details of the ADB I/O. The default hardware configuration is all channels of port A configured as digital outputs and all channels of port B configured as digital inputs. We match the default hardware configuration by calling the message handler configurePort for port A and port B. If you look at configurePort you see that it calls ConfigureADBIO and that if it succeeds it sets either the property pChannelTypesAList or pChannelTypesBList to the requested channel types. If you need other settings for either

GOT BUGS?

...and you're drowning in paper trying to keep track of them?

You need **BugLink!**

- **BugLink is a client/server application** allowing you to connect developers, testers, and support engineers anywhere in the world.
- **Intuitive user interface** gives you the information you need at a glance.
- **Fully customizable database**—add the fields you need to each project.
- **Custom TCP/IP protocol minimizes network traffic**—ideal for dial-up connections.
- **Client applications can operate 'off-line'**—allowing bug entry and modification even when not connected to the network!
- **Cross platform**—set up mixed Macintosh and Windows environments in minutes with no additional software needed!
- **Try before you buy.** Download and try risk free for 30 days from http://www.pandawave.com/bl/

A 5 User License starts at $299; that's only $60 per person.

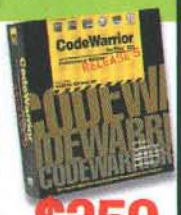
The Panda*Wave*

http://www.pandawave.com

The products you need, with the prices and service you deserve... guaranteed.

Power Tools for Programmers!

CodeWarrior Pro 5



CodeWarrior Professional 5 put everything you need for software development at your fingertips: project management tools, text and resource editors, source and class browsers, compilers, linkers, assemblers, and debugger.Release 5 offers such features as RAD for Java, faster compile times, local and remote application debugging, IDE extensibility options and even tighter C/C++ compliance. Additionally, you can create applications for Windows 95/98/NT and Mac OS 8.x and Mac OS X from either host platform. (available in versions hosted on Mac or Windows).

$359

Resorcerer 2.2

Resorcerer is the only supported general-purpose resource editor for Macintosh. Relied upon by thousands of Mac developers, Resorcerer features a wealth of powerful yet easy-to-use tools for easier, faster, and safer editing of Macintosh data files and resources. Whether you have to parse a picture, debug a data fork, design and try out Balloon Help, create a scripting dictionary, create anti-aliased icons, design and edit a custom resource with 40,000 fields in it, create C source code to run a dialog, or any of hundreds of other resource-related tasks, Resorcerer's magic will quickly save you time and money.

$256

Spotlight

Spotlight is the first Macintosh "Automatic Debugger". It can automatically locate run time errors in your code and display the offending source code line. Unlike similar tools on other platforms Spotlight is easy to use. No source code changes are necessary for application debugging. Spotlight can automatically check for wild pointers, memory leaks, overwrites, underwrites, invalid dereferencing of handles, and even toolbox parameter validity checking -- spotlight knows Macintosh verifying parameters to over 400 toolbox calls.

$189

VOODOO Server

VOODOO Server is a version control system for software developers using Metrowerks CodeWarrior under Mac OS. VOODOO Server and the corresponding VOODOO clients (the included CodeWarrior VCS plug-in and the VOODOO Admin application) are designed to offer reliable and robust version control features while minimizing the administrative overhead that usually accompanies version control. If you're a single programmer or managing a team of developers, version control can make or break your project. Do it right, with VOODOO

$199

BugLink Solo

BugLink Solo is a database dedicated to collecting and tracking problem reports. Ideal for the shareware developer. BugLink Solo allows you to distribute with your software BugLink Reporter, which can easily be configured to allow your end users to send structured bug reports directly to BugLink Solo. Promt users for exactly the information you need, send immidiate report conformations, customize a new database to each project, search for related bugs with a simple query. To kill bugs dead, you have to know where to find them.

$79

Developer's C++ Kit Professional

C++ Developer's Kit Standars was developed for everyone who, even without much knowledge of C++ language or software developing, wants to know the C++ programming language with applications in the OOSE. C++ Developer's Kit explains software solutions for C++ developers, programmers and C++ students.

$189

Page Charmer 2.0 $139

Scripter 2.0 $179

WebTen $349

ToolsPlus Lite $99

FaceSpan 3.0 $179

WebSpice 1,000,000 $89

WebSpice Animations $89

MkLinux $39

PowerKey Rebound $89

Developer DEPOT®

PO Box 5200 • Westlake Village, CA • 91359-5200 • Voice: 800/MACDEV-1 (800/622-3381)
Outside US/Canada: 805/494-9797 • Fax: 805/494-9798 • E-mail: orders@devdepot.com

www.devdepot.com

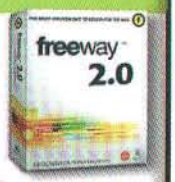
Master the Web!

WebSTAR Server Suite 4.2

WebSTAR Server Suite is a complete set of powerful and easy-to-use INternet servers for the Mac OS. Effortlessly serve web pages, host email accounts, publish databases, and share files - all with a single application on one Mac! WebSTAR Server Suite is perfect for Internet or Intranet serving, single or multiple sites, smalll and large businesses - it's power and ease-of-use saves any organization time and money.

$499

Freeway 2.0



Award-winning Freeway 2.0 is simply the fastest and easiest way to create web pages. Professional web designers worldwide use Freeway with its support for CSS and HTML 4.0, along with auto image slicing, image optimization, multilingual spell-checking, HTML Import, built-in FTP, and an exciting new Actions technology to create and maintain leading-edge websites - all without coding a single line. Freeway ships with a 390-page manual (5/5 - Computer Arts Magazine), Filemaker tutorial, a PhotoDisc offer with free images and more!

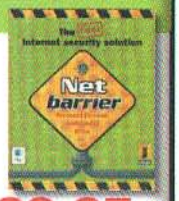
$199

Funnel Web Pro

Funnel Web is the ultimate web analysis solution for professionals. Specifically designed for profiling web site usage and monitoring customer usage patterns, Funnel Web is ideal for examining server performance and online effectiveness. Funnel Web can analyze log file formats from any server, WebSTAR, WebTen, even Unix or NT hosted ervers. Discover the most popular pages on your site, track server loads & optimize server performance, profile visitors based on organization, domain name, country, browser, etc. Funnel Web does it all!

$449

NetBarrier



NetBarrier offers a Personal Firewall, Antivandal protection, and Internet Filtering. Protect your machine from intrusions by Internet or AppleTalk. Incorrect passwords and individual actions are logged, you are alerted to hostile actions, and intruders are easily isolated. Internet Filtering allows you to be sure that passwords, credit card numbers, and other sensitive information can never be exported from your computer -- the content itself is filtered before any transfer! (Rated 4 mice by Macworld Magazine)

$69.95

...and great hardware solutions!

2 USB PCI Card

Add two USB ports to your older Macintosh. Connect up to 127 devices to the Universal Serial Bus (USB) that is Apple's new standard for desktop connectivity. USB mouse devices, keyboards, joysticks, game controllers, printers, scanners -- connect them all to your current computer. Installs in minutes!

$35.95

Swann MacSurfer USB JoyStick

Any serious gamer knows the superiority of the flight joystick. But until now, Mac users have been hard pressed to find a gaming joystick with the kick-butt power and resiliance they need. The MacSurfer USB JoyStick fits that gap with features like 4 trigger buttons, rapid fire finger trigger, hat control, throttle, two way self centering, and complete customization for your game with the included Overdrive software.

$29.95

Unreal $29.95

Lode Runner 2 $24.95

Klingon Honor Guard $39.95

Dark Vengeance $34.95

Myth II Soulblighter $39.95

Tomb Raider II $19.95

Deadlock $29.95

Future Basic 3 $169

CodeWarrior Discover Programming $49

port A or B you can call always call configurePort with the desired channel types after the child object has been created.

Note the flow of error messages: if ConfigureADBIO returns an error message, then configurePort returns an error message and the birthing handler returns 0. Also note the use of the property pDEBUGMessage. If it is true we print some error messages to the message window. ConfigureADBIO is pretty important. During debugging we need to know if it fails and why.

Reading from inputs

We will now look at the calls for getting information from channels and ports. There are four calls: getChannelDigital, getChannelAnalog, getPort and getPortDigital. Let's start with getChannelDigital (Listing 2).

Listing 2

getChannelDigital

This handler returns the value of a digital input channel.
Example: getChannelDigital (oADB1,"A", 1)
Possible return values: 0 (low), 1 (high) or an error string (failure).

```
on getChannelDigital me, iPort, iChannel

  — Make sure iChannel is within range 1-4.
  if iChannel<1 OR iChannel>4 then
    set lResult = "Error : Invalid Channel parameter"
    return lResult
  end if

  — Get the channel types and values for this port.
  put GetADBIO(pUnit, iPort) into lResult

  if word 1 of lResult = "Error" then
    — GetADBIO returned an error.
    return lResult
  else
    — GetADBIO returned no error.
    — Extract the relevant line for the desired channel.
    put line iChannel of lResult into lThisChannelString

    — Check whether whether our channel is a digital input
    — that is high, low, or not a digital input at all.
    case lThisChannelString of

      "digital in,high":
        — This is a digital input and it is high.
        return 1

      "digital in,low":
        — This is a digital input and it is low.
        return 0

      otherwise:
        — This channel does not allow digital input.
        set lResult = "Error: Channel" && iChannel &&¬
                      "of port" && iPort && "of unit" &&¬
                      pUnit && "is not a digital input."

        return lResult

    end case
  end if
end
```

If you just want to look at a single digital input channel, getChannelDigital is the way to do it. First it makes sure iChannel is within the range 1-4, then it calls GetADBIO to retrieve an up-to-date channel record for the port you specify in the iPort parameter. Based on the iChannel parameter it extracts the relevant line from the channel record. It then determines whether the channel is high, low or not a digital channel at all. If the channel is high, getChannelDigital returns 1, if it is low it returns 0. If the channel is not a digital input channel, getChannelDigital returns an error message.

The next call we examine is getChannelAnalog (Listing 3).

Listing 3

getChannelAnalog

This handler returns the value of an analog input channel. Since analog input is only possible on port B, we do not need to pass the port as a parameter.
Example: getChannelAnalog (oADB1, 1)
Possible return values: 0-255 (success) or an error string (failure).

```
on getChannelAnalog me, iChannel

  — Make sure iChannel is within range 1-4.
  if iChannel<1 OR iChannel>4 then
    set lResult = "Error : Invalid Channel parameter"
    return lResult
  end if

  put GetADBIO(pUnit, "B") into lResult

  if word 1 of lResult = "Error" then
    — GetADBIO returned an error.
    return lResult
  else
    — GetADBIO returned no error.

    — Extract the relevant line for the desired channel.
    put line iChannel of lResult into lThisChannelString

    — Check whether analog in is allowed on this channel.
    if lThisChannelString contains "analog in" then
      — This channel does allow analog input.
      — Extract its value.
      put char 11 to length(lThisChannelString)¬
      of lThisChannelString into lValueString

      return value(lValueString)

    else

      — This channel does not allow analog input.
      set lResult = "Error : Channel" && iChannel &&¬
                    "of port B of unit" && pUnit &&¬
                    "is not a analog input."

      return lResult

    end if
  end if
end
```

getChannelAnalog is the analog equivalent of getChannelDigital: it is the way to go if you need to monitor a single analog input channel. First it makes sure the parameter iChannel is within the range 1-4, then it calls GetADBIO to retrieve an up-to-date channel record for the port you specify in the iPort parameter. Based on the iChannel parameter it extracts the relevant line from the channel record. It then determines whether the channel allows analog input. If the channel is an analog input getChannelAnalog extracts the channel value. If the channel is not a analog input channel getChannelAnalog returns an error message.

But what if you need to monitor multiple channels? Calling getChannelDigital or getChannelAnalog repeatedly is not very efficient, as each time you call them, you in fact call GetADBIO which returns a channel record with the values of all the channels of a port. Reading from or writing to a port too often

can make other input devices on the ADB bus act sluggishly (Beehive Technologies recommends not to read more than 12 times per second and not to write more than 6 times per second). A more efficient way to look at multiple channels is by means of getPort (Listing 4).

Listing 4

getPort

This handler gets both the channel types and the channel values of the inputs of a particular port. The channel types are stored in pChannelTypesAList or pChannelTypesBList, depending on the port. The channel values are stored in pChannelValuesAList or pChannelValuesBList, depending on the port.
Example: getPort (oADB1, "A")
Possible return values: 0 (success) or an error string (failure).

```
on getPort me, iPort

  set lResult = GetADBIO (pUnit, iPort)

  if word 1 of lResult = "Error" then
    -- GetADBIO returned an error.
    return lResult
  else
    -- GetADBIO returned no error.
    -- Work out the channel value and the channel type.
    repeat with lChannel = 1 to 4

      set lThisChannelString = line lChannel of lResult

      case (lThisChannelString) of
        "digital in,low":
          set lChannelType = "digital in"
          set lChannelValue = 0

        "digital in,high":
          set lChannelType = "digital in"
          set lChannelValue = 1

        "digital out,low":
          set lChannelType = "digital out"
          set lChannelValue = 0

        "digital out,high":
          set lChannelType = "digital out"
          set lChannelValue = 1

        "analog ref,255":
          set lChannelType = "analog ref"
          set lChannelValue = 255

        otherwise:
          set lChannelType = "analog in"
          put char 11 to length(lThisChannelString) of¬
          lThisChannelString into lChannelValueString
          set lChannelValue = value(lChannelValueString)

      end case

      -- Store the channel type and channel value in
      -- their respective arrays.
      if (iPort = "A") then
        setAt pChannelTypesAList, lChannel, lChannelType
        setAt pChannelValuesAList, lChannel, lChannelValue
      else
        setAt pChannelTypesBList, lChannel, lChannelType
        setAt pChannelValuesBList, lChannel, lChannelValue
      end if

    end repeat

    return 0

  end if
end
```

In getPort we finally make use of the properties pChannelValuesAList and pChannelValuesBList. getPort calls GetADBIO for the desired port. If GetADBIO fails, getPort returns an error message. If getPort executes successfully and returns a channel record, getPort looks at each line of the channel record, puts the channel type in either pChannelTypesAList or pChannelTypesBList, and the channel value in either pChannelValuesAList or pChannelValuesBList. In the process the values of the input channels are converted to integers: 0 or 1 for a digital input and 0-255 for an analog input. After calling getPort you can easily retrieve an input channel's value by using the standard Lingo list access call getAt. You will see how to use the getPort-getAt combo in more detail when we put the parent script into action.

We don't discuss GetPortDigital in detail here. If you look at the code, you will see that it is simply a wrapper for GetADBIO with use of the number parameter. It reads all four channels of a port, converts the binary pattern on the port to a decimal number and returns this number.

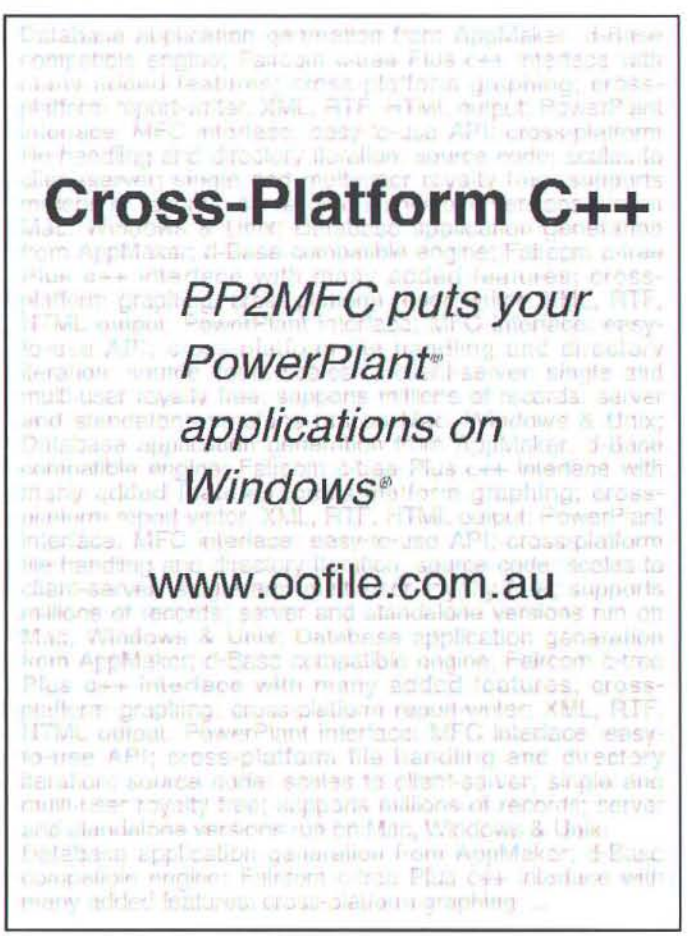
Writing to outputs

There are three calls for setting output channels and ports: setChannelDigital, setChannelDigitalDuration and setPortDigital. Let's start with setChannelDigital (Listing 5).

Listing 5

setChannelDigital

This handler sets a digital output channel high or low.
Example: setChannelDigital (oADB1, "B", 1, 1)
Possible return values:
0 (success) or an error string (failure)



```
on setChannelDigital me, iPort, iChannel, iChannelValue

  -- Check whether the requested channel value is allowed.
  if iChannelValue<>0 AND iChannelValue<>1 then
    -- The requested value is not allowed.
    set lResult = "Error: channel value must be 0 or 1"
    return lResult
  end if

  -- Try setting the channel to the requested value.
  if iChannelValue=0 then
    set lResult = SetADBIO(pUnit, iPort, iChannel, "low")
  else
    set lResult = SetADBIO(pUnit, iPort, iChannel, "high")
  end if

  return lResult

end
```

setChannelDigital is a convenient way of setting a single digital output channel. First it checks whether the requested channel value is 0 or 1. If the channel value is not 0 or 1, setChannelDigital returns an error. Otherwise it uses a call to SetADBIO to try to fulfill the request.

We are not going to cover setChannelDigitalDuration or setPortDigital in detail. If you look at the accompanying code, you will see that setChannelDigitalDuration is similar to setChannelDigital. The main difference is that the former uses the Time parameter in SetADBIO to set a digital output channel for a certain duration. This feature is available on port A only. With SetPortDigital you can set a whole port in one go, providing all the channels are digital outputs.

Taking It All for a Spin

Ok, let's try to put our parent script to the test. If you get stuck in this section, you may wish to peek ahead at the trouble shooting section towards the end of this article.

External electronics

To test the parent script you need to hook up some external electronics to the ADB I/O (Note that according to the ADB I/O manual, your Macintosh should be turned off before you attach or disconnect electrical devices from the ADB I/O). Connect an LED to each channel of port A. Hook up potentiometers to channel 1 and channel 2 of port B. Connect an LED to channel 3 of port B. Finally, connect a push-to-make switch to channel 4 of port B, which connects the channel to ground when you press the switch. If you feel unsure about how to connect these components, have a look at the application notes on the Beehive Technologies website.

Initialization and configuration

Have a look at the startMovie handler in the movie script of testADBIOMovie.dir. There are only three lines of code:

```
global oADB1
openXLib "ADB I/O XCMDs 1.0.1"
set oADB1 = new(script "ADB I/O", 1, TRUE)
```

The three lines should look familiar to you from the testOOMovie.dir and the testXCMDsMovie.dir. The first line creates a global variable called oADB1. The second opens the library with the XCMDs. The third line assigns to the global variable oADB1 a child object based on the ADB I/O parent script by calling the birthing handler. As iUnit parameter we pass 1 (the unit ID number) and as iDEBUGMessage we pass true.

Before we start playing around with the handlers remember that we took the pessimistic approach to debugging messages: we only get to see bad news. No message is good news. Now open the message box, delete any text that litters it, and play the movie. If the message box remains empty that is good news: the birthing handler and the embedded calls to configurePort for port A and port B executed successfully. If you do get an error message you should be able to figure out what is wrong. After all, we carefully passed back the original error message from ConfigureADBIO.

We can still get some kind of confirmation that the birthing handler executed successfully. Let's look at the channel types of port A by typing:

```
put the pChannelTypesAList of oADB1
```

The message box responds with:

```
-- [ "digital out", "digital out", "digital out",¬
    "digital out"]
```

That is good. It is default hardware configuration for port A which we requested in the birthing handler. Now look at the channel types of port B. Type:

```
put the pChannelTypesBList of oADB1
-- [ "digital in", "digital in", "digital in",¬
    "digital in"]
```

Again, that is what we expected. It is the default hardware configuration for port B which we requested in the birthing handler.

From the electronic components you can infer how we want the ports to be configured. All channels on port A need to be digital outputs. On port B, channels 1 and 2 need to be analog inputs, channel 3 needs to be a digital output and channel 4 a digital input. Since the desired configuration of port A matches the default configuration we do not need to reconfigure port A. The desired configuration of port B differs from the default configuration so we call:

```
put configurePort (oADB1, "B",  "analog in", "analog in",¬
                                "digital out", "digital in")
```

The message box should respond with 0 indicating that no errors occurred.

Reading from inputs

We will cover the remaining handlers in the order they occur in the parent script. Let's try reading channel 4 of port B, a digital input.

```
put getChannelDigital (oADB1, "B", 4)
```

Providing the jumper on this channel is set to pull-up and you use a push-to-make switch connected to ground, getChannelDigital returns 1 with the switch open and 0 with the switch closed.

Programming Doesn't Have To Be This Difficult. It's REALbasic!

REALbasic is the award-winning, visual, object-oriented BASIC development environment for the Macintosh.

Our application is easy enough for beginners and powerful enough for professionals to use. Whether you want to learn *(or return!)* to programming, REALbasic is for you!

Simply drag and drop user interface elements into your application's windows and dialog boxes to create anything from a small utility to a large application or the next great game—faster and easier than ever before. Experiment, explore, learn, and innovate as you build your applications step by step while REALbasic handles all the details for you so you can concentrate on what makes your software great—your ideas!

Best New Product

Macworld

MacHome

With the Professional Edition you can even access most databases and create applications that run on Windows.

While REALbasic is easy to use, it's also extremely powerful. Features like multi-threading, classes, inheritance, a powerful debugger, extensibility, VisualBasic project import, advanced dynamic recompiling compiler, TCP/IP controls, and multimedia tools make REALbasic the perfect combination of simplicity and sophistication.

Go to www.realbasic.com NOW to download a FREE trial version or call 512.263.1233.

REALbasic™

REALbasic is a trademark of REAL Software, Inc.

Next we try reading an analog input, channel 1 of port B.

```
put getChannelAnalog (oADB1, 1)
```

getChannelAnalog should return a value. Now turn the potentiometer connected to this channel and call getChannelAnalog again. Notice how the value of the channel changes. Now try getChannelAnalog with channel 2 of port B. It should behave in the same way.

The last call for reading inputs is GetPort. Let's call it for port B:

```
put GetPort (oADB1, "B")
```

Remember that GetPort stores the channel values of a port in either pChannelValuesAList or pChannelValuesBList. Let's have a look at the property pChannelValuesBList.

```
put the pChannelValuesBList of oADB1
```

The message box shows a list with four values which correspond with the four channels of port B. We can simply get at each entry in the list through the use of getAt.

```
put getAt (the pChannelValuesBList of oADB1, 1)
```

gives us the value of the analog input channel 1 of port B. Likewise,

```
put getAt (the pChannelValuesBList of oADB1, 2)
```

gives us the value of the analog input channel 2 of port B. And

```
put getAt (the pChannelValuesBList of oADB1, 4)
```

gives us the value of the digital input channel 4 of port B.

List access calls to pChannelValuesAList or pChannelValuesBList must be preceded by a call to getPort. If you do not call getPort first, the lists contain old values which may differ from the current values. Think of getPort and getAt as a combo.

The last of the message handlers for reading from the ADB I/O is getPortDigital. With the components we have got hooked up, we can't really try it out in a meaningful way which would be to convert the binary pattern on a port with four digital input channels to a decimal number. What we can do is show its use with the four digital output channels of port A. First set the port with the XCMD SetADBIO, then read it again with getPortDigital.

```
SetADBIO 1, "A", 15
put getPortDigital (oADB1, "A")
```

If you play about with these commands, you will find that you can read back from the port the value you just wrote to it. Note that if you use getPortDigital with a port that has one or more channels set to analog in, you will not get an error message. An analog input channel will have the value 0. While that is not useful in itself, it allows you to work out the values of any remaining digital channels.

Writing to outputs

Luckily, writing to channels and ports is a bit easier than reading from them. Let's write to channel 3 of port B, a digital output. Typing

```
put setChannelDigital (oADB1, "B", 3, 0)
```

turns off the LED.

```
put setChannelDigital (oADB1, "B", 3, 1)
```

turns on the LED. Try using setChannelDigital on port A.

Next, try setChannelDigitalDuration. Remember, this call only works on port A. Let's try it on channel 1 of port A.

```
put setChannelDigitalDuration (oADB1, 1, 1, 10)
```

This turns on the LED for 10 x 0.1 = 1 second after which the LED turns off.

Finally, there is setPort for setting a whole port in one go. Enter

```
put setPort (oADB1, "A", 15)
```

This turns on all LEDs on port A, since the binary equivalent of the decimal integer 15 is 1111.

Play around a bit more with all these handlers. Try forcing some errors. For example, try writing to an input channel or reading from an output channel. Error messages will inform you if you ask for the impossible.

Trouble Shooting

Strange bugs but no error messages? Or error messages which completely puzzle you? This section lists some possible causes.

Software

1. Have you enclosed the parameters in brackets? All our message handlers are in fact functions, that is, they return values. Lingo expects functions to have their parameters enclosed in brackets.

2. Do the values you pass into the handler match the handler's function prototype? If they do not, confusing error messages may result.

3. Are all child objects of the ADB I/O parent script declared as globals? If you expect a child object to stay around in memory, while it is actually a local child object which goes out of scope at the end of the handler in which it was created, you can get very confusing errors. If in doubt try using put to print the value of the child object to the message box. If you get void you are half way to finding the bug.

Hardware

1. Is your port A channel correctly equipped with a relay or an opto-isolator? If you need digital output on a port A channel you need to have a relay installed, if you need digital input you need an opto-isolator installed. Obviously,

if you have a relay installed on a channel and try to configure it as a digital input, or have an opto-isolator installed and try to configure it as a digital output, that channel will not work correctly. But as the processor in the ADB I/O cannot distinguish between relays and opto-isolators ConfigureADBIO will not return an error string.

2. Is your port B channel jumper correctly set to pull-up, pull-down or open circuit? If you need digital input or digital output on a channel of port B, you need to set the jumper to pull-up or pull-down. If you need analog input you need to disable the jumper (open circuit). If you try analog input with pull-up or pull-down you will get erroneous values, but you will not receive any error message.

Enhancements

To keep the number of files to a minimum I made the testADBI/OMovie.dir self-contained. You may find it convenient to put the parent script in an external cast, so that you can easily include it with different movies.

Another point which you may have noticed is that there is no analog output available on the ADB I/O. However, Application Note 6 on the Beehive Technologies website documents how to use both port A and B to drive four digital potentiometers to create four analog outputs. If you have a need for analog outputs you may wish to extend the ADB I/O parent script with a setChannelAnalog handler which considers each digital potentiometer to be a channel. The prototype would look like:

```
on setChannelAnalog me, iChannel, iChannelValue
```

Conclusions

In this article we used object-oriented Lingo to create a friendly and clean interface between the ADB I/O and your Director movie. Object-oriented Lingo allowed us to hide set-up details in the parent script's birthing handler and get rid of global variables through the use of property variables. If you had never done anything like this before, you don't know half how spoiled you are. Director and the ADB I/O form a truly Mac-worthy couple: this is data acquisition 'for the rest of us'. Have fun!

References

- Beehive Technologies website: <http://www.bzzzzzz.com/BeeHive/ADB_IO/ADBIO_Intro.html>
- Macromedia's website: <http://www.macromedia.com/software/director/>
- Griffin Technologies website: <http://www.griffintechnology.com/imac/imate.html>
- Small, Peter. *Lingo Sorcery : The Magic of Lists, Objects and Intelligent Agents.* 2nd edn. John Wiley & Son Ltd, 1999.

MT

BMS

THE LAW OFFICE OF BRADLEY M. SNIDERMAN

23679 Calabasas Road #558 Calabasas, CA 91302

Tel: 818-222-0365 Fax: 818-591-1038

Got Software?

Need help safeguarding your software? If you're developing software, you need your valuable work protected with copyright and trademark registration. Then, when you are ready to sell it, you can protect yourself further with a licensing agreement.

I am a California Lawyer focusing on Intellectual Property, Corporate, Commercial, and Contract Law, as well as Wills & Trusts.

Please give me a call or an e-mail. Reasonable fees.

The Law Office of Bradley M. Sniderman

Visa
Master Card

BMS

Discover
American Express

(818) 222-0365 • brad@sniderman.com

By Paul Shields

Backup Management

A guide to managing local and remote backups for your Mac

Introduction

When was the last time you lost data because of a hard drive failure? How about by accidentally deleting a file, whose name you did not recognize or felt you no longer needed? Have you ever lost data because of a fire or theft? All of these represent real situations that people face every day. How long before you encounter a similar situation?

Backups are one of the most critical, but often neglected, tasks in proper data management. Both home users and network administrators have lax attitudes towards backups until the day comes when they lose data, especially important business-related data. With the refinements in hardware and software technologies backups are easy and relatively inexpensive, even for the small office and home users.

How effective is a company where the list of customers is lost because of malicious activity by a former employee or Internet hacker? Can a company survive if its financial system lies in ruins because a hardware failure rendered it useless? As systems administrators your responsibility is to protect corporate computing assets. Since data is becoming one of the most valuable assets a company has, implementing a reliable backup system is essential. As a business owner, you must take interest in the processes used to protect your assets. There is no excuse for being caught off-guard.

Backup Terminology

Before beginning a discussion on implementing a backup system, we must define some of the common terms.

Full backup

A full backup is a complete copy of everything on the source drive to the backup media. Most users and administrators run a full backup on a weekly or monthly basis.

Incremental backup

An incremental backup only copies the data that has changed since the last backup. Administrators will run incremental backups on a daily basis to capture the data files added or updated during the day.

Differential backup

A differential backup copies the data that has changed since the last full backup. It is important to understand the difference between an incremental and differential backup. Administrators use differential backups to facilitate more complex tape rotation schemes and sometimes in database environments.

Archiving

Too many administrators fail to understand the difference between archives and backups. They fail to see the powerful capabilities of archiving and use their normal backups as archival storage. Archiving is the process of moving data from one media (typically a hard drive) to another media (optical disks or tapes). Archiving deletes the files from the original location. Companies may use archiving to store things such as historical financial data. Archiving saves space on hard drives, while retaining the ability to restore the data quickly.

Paul Shields is currently the Technology Advisor and Project Manager for a major telecommunications firm in Dallas, TX. In his role Paul selects and implements the critical technologies that comprise the backbone of the computing infrastructure. He also writes for several publications including AppleLinks http://www.applelinks.com/ and develops disaster recovery plans that minimize downtime and maximize recoverability. Feel free to forward any questions or comments to him at pshields@cyberramp.net.

Picking the Right Media

Each media type has dramatically different characteristics including lifespan, storage capacity, re-usability, and costs. Even within a category such as tape drives the number and variety of options available is broad enough to suit a wide spectrum of users. One important factor in selecting a media is the length of time you plan to store the media. Table 1 provides a lifespan comparison of some of the most popular backup media.

Media	Lifespan (years)
Magnetic Tape	1 - 3 (newer tape technologies are extending this)
Magnetic Disks	3 - 5
Optical Disks	30
Write-once CD-ROM	30+

Table 1. *Average lifespan of common backup media. This is strictly a shelf-life and does not factor in the number of uses.*

A common mistake is to pick a media type based on the needs of today's systems. The data requirements of businesses are growing at a phenomenal rate estimated to be as high as 70% per year by the Gartner Group. A system that barely meets the capacity and performance requirements of your current systems will be inadequate within 18 months and the search will be on for a replacement. Switching media types is an expensive and time-consuming proposition.

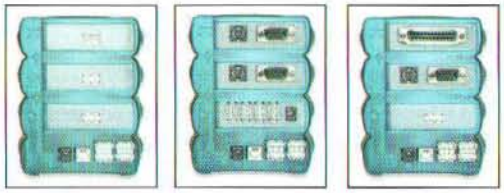
Tape drives

Tape is the standard in the backup systems of the majority of corporations. Tape offers extremely high capacities, excellent performance characteristics, and a reasonable lifespan. The variety of formats makes tape a versatile media that can fit in a number of environments. Table 2 provides a summary of the most common tape types and their specifications.

Type	Capacity (uncompressed)	Speed (uncompressed)	Drive Costs	Media Costs
QIC (Travan)	TR-4: 4 GB TR-5: 10 GB	25 MB/min 60 MB/min	$350 $550	$7/GB $4/GB
DAT	DDS-2: 4 GB DDS-3: 12 GB DDS-4: 20 GB	46 MB/min 60 MB/min 180 MB/min	$650 $1050 $1350	$2/GB $2/GB $2.50/GB
DLT	DLT 4000: 20 GB DLT 7000: 35 GB	90 MB/min 300 MB/min	$2000 $4000	$4/GB $2.50/GB
AIT	AIT: 35 GB AIT II: 50 GB	180 MB/min 360 MB/min	$2000 $4000	$3/GB $2.75/GB

Table 2. *Tape types and specifications. Most drive manufacturers offer compression and advertise compressed storage capacities that are roughly double their uncompressed capacities.*

Imagine that. Smart and beautiful.



- Seven port hub (standard configuration)
- Two Serial and eight USB ports
- One SCSI, one serial and five USB ports
- Sixteen USB ports
- One serial and twelve USB ports
- One SCSI, two serial and four USB ports

Think about it.

Why should you have to buy a USB hub, then hang stuff off of it until it looks like something out of *Edward Scissorhands*? Why fumble around under your desk every time you want to plug in another device? Instead, try this: A modular, stacking system of USB hubs, with adapters for serial, SCSI and more. Neat. Compact. And, oh yeah... drop dead gorgeous.

BELKIN

PID Participants

COMPUSA fry's Insight MacConnection MacMall macZone MICRO CENTER MicroWAREHOUSE

©2000 Belkin Components. All Rights Reserved. All Trade Names Are Registered Trademarks Of Respective Manufacturers Listed. 20AD183

BELKIN

usb.belkin.com

Belkin Components
Compton • CA
310.898.1100 • Fax 310.898.1111
United Kingdom • Holland • Atlanta, GA

For a home or small business user, the low cost of the Travan drives makes them an appealing option. The catch is the high media costs per GB and the slow performance. Although 60 MB/min sounds fast, a typical hard drive is now 4 - 8 GB, which translates into a full backup time of 1 - 2 hours. This long backup time makes it a nuisance and you are less likely to be anxious to give up access to your machine for a few hours while it backs up.

For a large organization, picking the right technology depends on the type of backups. Most administrators deploy DAT tape drives for network backups because they are less expensive and offer good performance and capacity. AIT and DLT offer more capacity and better performance but are expensive. DLT and AIT are perfect for situations where you attach the tape directly to a fileserver because of the large amounts of data on the server.

The reason DLT and AIT may not fit well in a typical network backup is the potentially slow performance of moving data between machines. The latest releases of Retrospect and OS 8.5 combined with a quality TCP/IP network can come close to the speeds needed to use the full capacity of a DLT or AIT system.

To find out why DAT may be better for network backups than DLT compare the difference in performance between a 10Base-T network and DLT drives. At best, a 10Base-T network can move data at about 1 MB/s. A DLT writes data at a minimum of 1.8 MB/sec. This gap is even wider with the new AIT II tapes and the proposed SuperDLT format. DAT drives more closely match the performance of a 10Base-T network and have a lower cost per MB. DLT is still viable and I use it for network backups on a regular basis, primarily because of its longer lifespan and better durability.

Another reason to choose tape as a backup media is the availability of tape libraries. A tape library is a device that contains one or more tape drives along with dozens or hundreds of tape slots. These systems have a robot that can automatically swap tapes as directed by the host computer. Tape libraries greatly enhance the storage capabilities of a backup server. No longer do administrators have to limit backups to what will fit on a single tape or constantly monitor the server and rotate tapes. The software automatically swaps tapes as they fill. Retrospect supports a number of tape library systems via the Advanced Driver Kit option.

Most tape drives have a SCSI interface, which poses a problem for iMac and B&W G3 owners without a SCSI card. Until Firewire or USB connections become more common on tape drives, users must consider alternatives or buy an adapter. USB is a low-speed interface and while adapters exist, the performance they offer will be disappointing. Firewire offers performance levels higher than traditional SCSI, so the use of an adapter should provide adequate performance. Tape drives with native Firewire should begin to appear in the next 6 - 12 months.

CD and DVD

Over the last few years, CD and DVD have become viable backup media for small businesses and critical data in large businesses. As shown in Table 2, both CD and DVD drives offer good storage capacity and performance with moderate media costs.

Capacity (uncompressed)	Speed	Drive Costs	Media Costs
CD	650 MB	600 KB/sec	$379	R: $1/GB
RW: $10/GB			
DVD	5.2 GB (2.6 GB/side)	1300 KB/sec	$599	$8/GB

Table 3. *CD and DVD specifications. These media types offer smaller capacities than tape but have other advantages that make them appealing for archival purposes.*

The real advantage of a CD or DVD is the long lifespan of the media. Optical media can last as long as 30 years before oxidation starts to affect the stored data. This makes optical media perfect for storing archives of critical data such as financials or personnel records.

One note of caution when considering DVD is that the capacity is 2.6 GB per side. Switching sides on the DVD recorder is a manual process, which interferes with scheduling unattended backups.

Removable drive technologies

Zip drives are almost universally available and are relatively inexpensive. Jaz drives offer more capacity and better performance while retaining a relatively low cost. These media though do not offer extremely high levels of reliability and are best suited for file transfer, occasional ad-hoc backups, and short-term storage.

Another problem with removable media is the high costs of media. A typical Jaz cartridge costs $90 and stores 1 GB of data. A typical hard drive now tends in the 6 GB range. If we assume the hard drive is full and that you run a full backup once per week, then the yearly cost approaches $28,000. While you may be able to reduce this cost slightly by recycling cartridges, these costs do not account for having duplicates of your backup media (highly recommended) nor the media used for daily incremental backups. This compares poorly to DAT, which under the same conditions has a yearly cost of just over $1000.

One promising technology is the Orb drive from CastleWood <http://www.castlewood.com/>. The Orb uses traditional hard drive media and offers the potential for improved reliability as compared to the Zip/Jaz technologies. EIDE versions are shipping now, while SCSI, USB, and Firewire version should be available by the middle of the year. The Orb solves some of the media cost problem by reducing media costs to approximately one-third the cost of Jaz media. While this difference does not make the Orb a competitor to tape in most situations, it may make it acceptable in some circumstances.

Which one is right for your needs?

Most users should avoid removable magnetic drives because of their limited capacity and short lifespans. A small

business with a few machines could probably do well with the older DAT technologies (DDS-2 or DDS-3). Both offer good performance and low media costs. The newer DDS-4 drives also support the older tapes, so when it comes time to upgrade there is an evolutionary path.

If you needs are beyond the capacity of DAT, both DLT and AIT are excellent solutions. DLT has been in the market place longer and is potentially a more stable technology. Sony is pushing the AIT standards hard though and the recent AIT II drives surpass DLT in performance, capacity, and media costs. A network administrator will not go wrong with either of these technologies, especially for backups of large data servers.

Media Rotation

Where to put your backup tapes?

Few administrators take the appropriate actions when it comes to storing their backup tapes. All too often when you ask the systems administrator where the backup tapes are, they point to a cabinet in their office. This is strictly a matter of convenience and complacency. Administrators hesitate to send the tapes off-site because as soon as they do, a user will request a file restore and they must bring the tapes back on-site. The complacency is because so few network administrators have been through a major disaster that resulted in a major loss. Few companies have a disaster recovery plan they enforce and there is little incentive for the network administrator to create one.

Home users and small business should invest in a fireproof safe at a minimum. A better solution is to find a secure off-site storage location like a safe deposit box at a local bank. Either way, the data is safe from the most common hazards such as fire, theft, or other unfavorable environmental conditions.

For large organizations, storing the tapes at a remote office or an off-site storage company is ideal. When selecting a site, ensure that the provider maintains proper environmental conditions and takes measures to control access to the backup media. Our local off-site storage company will not reveal its exact location under normal circumstance. They provide a courier pick-up and drop-off service for transporting the tapes between sites. We ship the tapes in sealed boxes to ensure that no one tampers with the contents.

Once you select a storage location, its time to decide how often the media will rotate off-site.

Developing a rotation schedule

The media rotation schedule forms the basis of your backup schedule. Its at this point that you start asking questions like, how much data can I afford to lose? There is no set standard for defining a schedule but there are a few guidelines. The exact schedule you choose will depend on the amount of backup data and the critical nature of the data.

The most common method is to do one full backup per week, usually over a weekend, and incremental backups the rest of the week. You have the option of placing the full and incremental backups on separate tapes, allowing you to ship the full backups off-site immediately on Monday morning. The incremental backups from the rest of the week are on a second set of media. You then ship the incremental tapes off-site on Friday before the next full backup. This gets the media off-site quickly and minimizes the amount of data at risk.

The problem with such an aggressive off-site storage policy is the inevitable restore request. The administrator must pull tapes back on-site to complete the restores. A potential solution is to bring the older backup tapes back on-site which may cover some restore requests and minimize the amount of tape shuffling.

Another option is to duplicate tapes before sending them off-site. While this method does double your media costs, it allows you to have one copy off-site for disaster recovery and one on-site for restores. To duplicate backup media you will need a machines with two drives. Retrospect has a feature to facilitate duplication of media.

The important thing is to keep the tapes off-site as much as possible. The whole purpose of backups is to provide safe and secure storage for critical data. Leaving them on-site means they are vulnerable to the same disasters that may destroy the original source.

Picking a Backup Package

There are few options when it comes to quality backup software for the Mac. Dantz Development's Retrospect is the standard by which we judge other packages. Three versions of Retrospect are available. The target customer for each package is different and selecting the one that meets your needs is relatively easy.

StoneTable

You thought it was **just** a replacement for the List Manager ?

We lied, it is **much** more !

Tired of always adding just one more feature to your LDEF or table code ? What do you need in your table ?

Pictures and Icons and Checkboxes ?
adjustable columns or rows ?
Titles for columns or rows ?
In-line editing of cell text ?
More than 32K of data ?
Color and styles ?
Sorting ?
More ??

How much longer does the list need to be to make it worth $200 of your time ?
See just how long the list is for StoneTable.

Make StoneTable part of your toolbox today !
Only $200.00 MasterCard & Visa accepted.

StoneTablet Publishing
More Info & demo Voice/FAX (503) 287-3424
http://www.teleport.com/~stack stack@teleport.com

Retrospect

Retrospect is the most comprehensive backup package for the Mac and will be the package administrators choose for their network backups. Retrospect can backup both local and remote drives. It also provides support for backing up Windows 95/98/NT machines from your Macintosh server. There is an add-on package, the Advanced Driver Kit, that adds support for high-end tape devices and tape libraries.

Retrospect Network Backup Kit

This package includes both the Retrospect server and Retrospect client software. This is the package network administrators will need in order to do centralized backups of all their machines using one or more servers. Each Retrospect client requires a unique license and additional licenses are available in 5, 10, and 50 packs.

Retrospect Express

Retrospect Express is what one might call the consumer version of Retrospect. The Express version lacks support for tape drives, but still has the ability to backup to local removable drives or over the Internet (via FTP). These features along with its ease-of-use and low price make it a package suitable for home users.

Backups for One

Local backups are best suited for home users and very small offices where there are only a few machines. In an office of dozens of users, there is a certain economy of scale to switching from local backups to a centralized dedicated backup server.

Many administrators will also use local backups for servers with large amounts of data to improve backup performance. Trying to backup several hundred gigabytes of data over the network is a time-consuming task that may not complete in a constrained backup window.

Configuring Retrospect for local backups

Retrospect offers an excellent utility called Easy Script to guide you through the process of building a backup script and schedule (**Figure 1**). Easy Script will automatically run the first time you launch Retrospect. Otherwise, it is available from the Help menu or the Scripts tab in the main window.

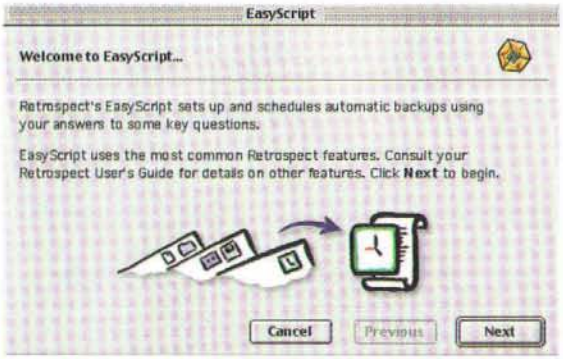


Figure 1. *Retrospect Easy Script utility for generating a backup script and schedule. Easy script is available from the Help menu or the Scripts tab.*

Individuals and network administrators will find the interface of Easy Script easy to use and thorough. Easy Script is part of both Retrospect and Retrospect Express. When you start Easy Script, it asks if you are configuring the script for local or network backups. The next step is to select the backup media (**Figure 2**).

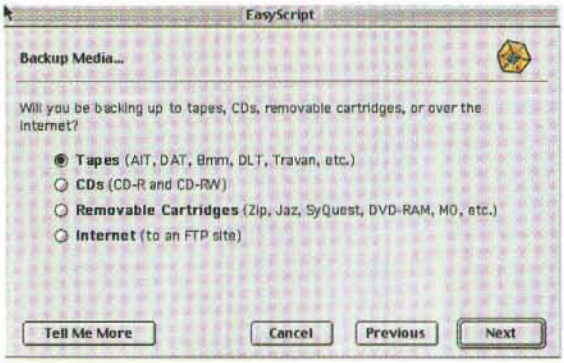


Figure 2. *Selecting the media type in Retrospect's Easy Script utility. Retrospect Express would not include the tape options.*

Each step in the Easy Script process includes additional information on the options available. After selecting how often you want to backup, Retrospect asks how often you want to rotate the media. The most commonly recommended pattern is to backup daily and rotate media on a weekly basis (**Figure 3**).

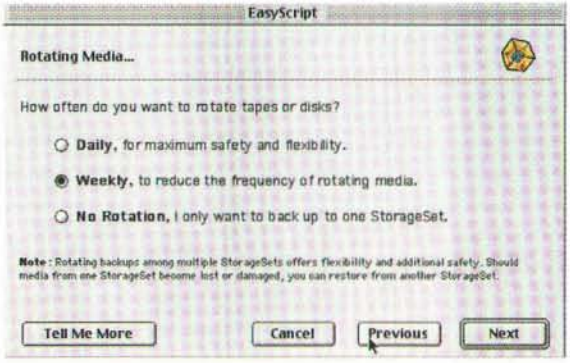


Figure 3. *Selecting the media rotation in Retrospect. While available, few should have a reason to select the no rotation option.*

After selecting the media rotation schedule, Retrospect provides a summary of the intended backup schedule and prompts the user to provide a preferred time for backups (**Figure 4**). When you click create, Retrospect will prompt you for the name of the storageset catalogs (**Figure 5**).

The default names are the word "Storageset" followed by a letter (A, B, C). While this naming convention works, it is not ideal. A more descriptive naming convention would be appropriate. One suggestion is to use the name of the server such as "Finance Backup Server A" and "Finance Backup Server B." These names indicate the server and backup rotation, making it easier to track which tapes go with each server. There is a 31-character limit because of filename restrictions in the Macintosh OS.

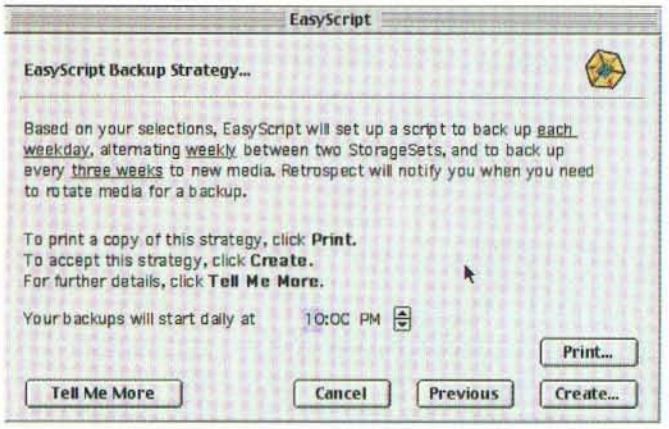


Figure 4. *Retrospect displays a summary of the backup schedule and prompts for a preferred backup time. At this point Retrospect will build the backup script.*

Figure 5. *The final step is to name the catalog files. Retrospect provides default names, but most administrators will prefer to set a more descriptive name.*

At this point, your backup server is ready to go and the only thing left is to load the media and wait for the next scheduled backup. Retrospect offers more flexibility to control the backup process, but most home users and small offices can ignore these options and use the default settings.

Backups for Everyone

Configuring Retrospect for network clients

The process of configuring the server to backup network clients is almost identical except for one step, the configuration of each network client. For network backups, you will need a unique license key for each Retrospect clients. License packs are available in units of 5, 10, and 50. Some OEM drive manufacturers will include a 3-user license with high-end DLT, DAT, and AIT drives.

The first step in setting up network clients is to install the Retrospect client control panel on each computer (**Figure 6**). The control panel is mostly non-obtrusive and in my experience has a tremendous record for stability and compatibility with both other extensions and OS updates. The name of the file has a special character at the beginning to force it to the bottom of the control panel's alphabetic list thus causing it to load last.

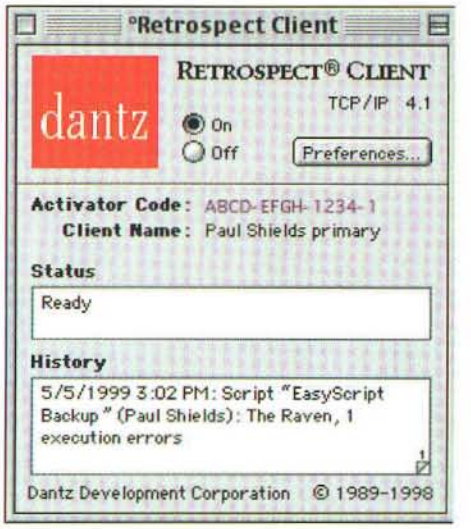
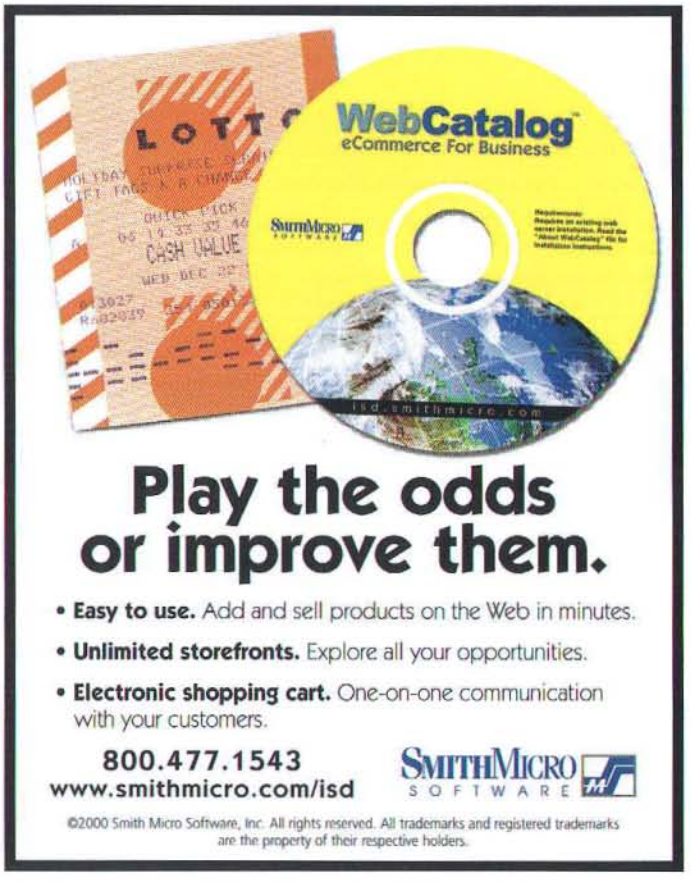


Figure 6. *The Retrospect Remote control panel on each network client is easy to install and has few compatibility issues.*

After installing the Retrospect client on each computer, you follow the same Easy Script process outlined for a single machine environment. The only change is the prompt to find and configure clients (**Figure 7**). Retrospect supports AppleTalk and TCP/IP connections to clients. When you first install the Retrospect client on a Mac, it will generally default to AppleTalk. Once captured



and configured in Retrospect, the administrator can change a client to use TCP/IP. Windows clients only support TCP/IP. The primary advantage of TCP/IP is a potential for a 2X increase in backup performance.

When you click on a zone, Retrospect sends out a broadcast to the zone looking for clients. If you select TCP/IP, the server will send out a broadcast to the subnet looking for clients. If you want to configure a TCP/IP client on another subnet, you will need to know its IP address or DNS name.

To configure the client, you double-click on it in the window and enter a license code. During the client configuration, Retrospect will also prompt you for a client name and password. Specifying a name allows an administrator to identify the machine, instead of relying on the Machine (File Sharing control panel) name or DNS name (PC clients). While you can choose to leave this password blank, doing so creates a major security hole in your backup system.

The password ensures that only authorized backup servers attach to the client and pull the client's data. By not setting a password, you leave the machine open to anyone on your network configuring a copy of Retrospect to capture and backup the data on the client. The person with the rogue backup server now has access to other people's data.

With a password configured, when a Retrospect server attempts to connect to the client for the first time, a password prompt will appear and the administrator must enter the correct password. The server will cache the password so that it can connect to the client for future backups without having to prompt the administrator.

For those administrators with sensitive data that requires an even higher level of security, Retrospect offers the option of encrypting the data before sending it across the network. This keeps someone from configuring a protocol analyzer on the network and capturing the data as it travels between client and server. On newer machines, the performance overhead of encryption is minimal.

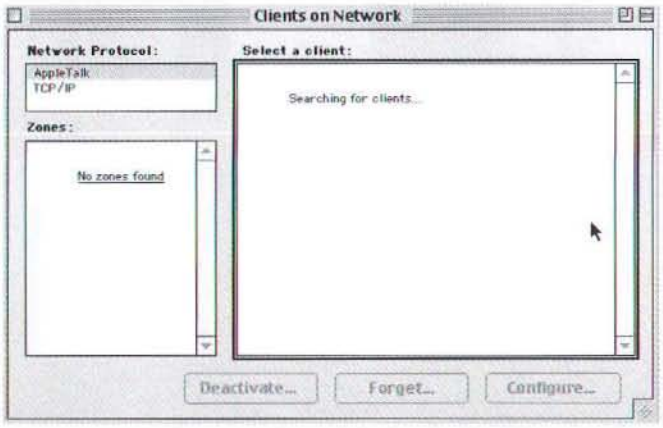


Figure 7. The interface for adding network clients supports both AppleTalk and TCP/IP clients.

After configuring each client, the process continues with the schedule and media rotation settings.

The problem with portables

The main issue when backing up clients over the network is finding an appropriate time to run the backups. Portable computers, which the users tend to disconnect from the network at night, complicate the issue. The most common mistake is to configure the client machines during the day when they are all on and connected to the network without realizing the machine is a portable. When backups run at night, the machine is gone.

The most common solution was to pick a time of day when there is a high probability of the machine being present on the network such as lunchtime and scheduling backups of portables to occur then. This method presents two basic problems. First, there is still no guarantee that the machine will be present on the network. Second, lunchtime varies from user to user and many may be in the middle of completing a critical task when the server starts a backup. No matter how fast the machine, backups can be an intrusive function that affects performance. In the past, we have found that the user reaction to such an immediate slowdown in performance is to restart the machine in the hopes that it would solve the problem. This results in the cancellation of the backup of that machine. The Retrospect client provides a partial solution to the problem, by letting a user lower the priority of the backups in the Retrospect control panel preferences. With the faster processors available today, this option can allow backups to proceed at almost full speed while maintaining usability for the user.

An administrator can carefully monitor the backup logs and attempt to contact the user in cases of repeated backup failures, but is a burdensome task. Contacting the user via the telephone can be just as difficult as finding their machine on the network. In addition, just because you manage to get hold of them, does not mean they will let you initiate a backup of their system immediately.

Retrospect 4.0 and later provides a solution to this problem that I have yet to find an equivalent of in any other software package. In version 4, Dantz introduced the concept of a backup server. Administrators configure backup server scripts the same as any other backup script, although you cannot use the Easy Script tool. The administrator selects the backup media, clients, storagesets, and schedule. A typical backup server script might look like the example shown in **Figure 8**.

The difference is that a backup server script runs continuously, constantly monitoring the network for clients. The backup server works through the list of clients and checks to see if the client is visible on the network. If the client is visible, the backup script checks

the amount of time since the last backup (a configurable option). If the client is due for a backup, then the server sends a notice to the client informing them that backup will begin in 20 seconds (another administrator configurable option). The user has a few options at this point. They can let the backup proceed, which is what happens by default or they can reschedule it for another time. If they choose to reschedule the backup, the server notes the requested date/time and makes an effort to retry the backups then.

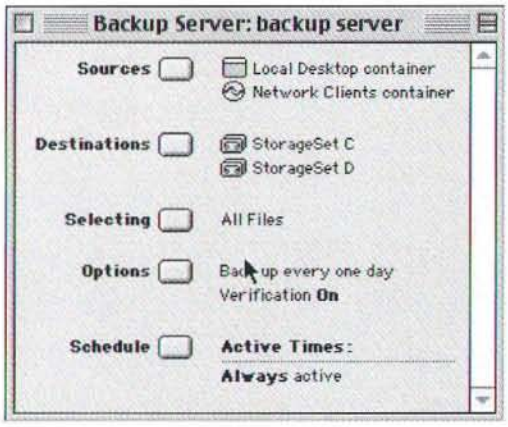


Figure 8. *Typical backup server script backs up all local and network drives to storageset C or D. This script is always active but will not backup a particular machine more than once a day unless the user requests it.*

Using the backup server script gives the user another option. Often a portable user will come in to the office for a short period to get a specific task done. If they know they will only in the office for a short time and want to ensure backups initiate, they can go to the Retrospect control panel and schedule a backup with the server. They can specify a certain time and date or simply tell the server to back them up ASAP. The server will move their machine to the top of the list and begin backups as soon as any in progress backups complete.

The only problem is explaining to the user the difference between ASAP and now. Many of them assume by clicking ASAP, they are telling the server to backup them up immediately. Since Retrospect can only backup one client at a time, the request goes into a queue. Depending on what stage the in-progress backup is at; this could mean a significant wait.

While the backup server option is infinitely better than the alternatives, it still has limits. Tape rotation is not an integral part of the backup server script. The backup server will use any tape that matches one of the configured storageset options. Administrators must be careful to load and name the tapes before the backup server script initiates. There is no way to code in the script to rotate tapes on a weekly basis, so inserting the wrong tape or using the wrong name may result in incremental backups using the wrong storageset.

A configuration that combines the power of regular backup scripts and the backup server might look like this. The administrator defines two scripts, one called Full backups and one called Daily backups. The administrator sets the full backup script to alternate between tapes on a weekly and runs every Saturday. The daily backups are a backup server script and run continuously Monday - Friday. The administrator inserts a blank tape on Friday evening which the full backup script sees and names. On Monday the administrator can leave the same tape in and store both the Full and Daily backups on the same tape. The other option is to remove the tape, insert a new blank tape and name it to match the storageset from the Full backup. This ensures that Retrospect bases the next incremental backups on the most recent full.

Summary

Backups are a critical but often neglected task. Proper backups are relatively easy to manage and the time spent in doing backups will be worth every minute the next time you or one of your users accidentally deletes a critical file.

There are three basic steps to developing a backup plan. First, select a media type that meets your storage and archival needs. For most home users, magnetic media or low-end tape drives will be adequate. Network administrators will want a high performance drive mechanism like DAT. DLT or AIT. If the goal is long-term archiving than CD-ROM or DVD is a perfect solution.

The second step is deciding on a media rotation schedule and planning a storage location for the backup media. You should make every effort to store backup media off-site such as in a safe-deposit box or other storage facility. While on-site, tapes should be in a fireproof safe. How often you rotate backup tapes between sites depends on the critical nature of the data and the number of restore requests you process on a daily basis.

The final step is configuring the software for your local machine or on a server that can handle backing up remote clients. Once implemented most backup systems are easy to maintain. Retrospect offers some excellent reporting capabilities that make it easy to monitor the backup status of all machines. This does not mean you can configure the system and ignore it. MT

By Neil Mayhew, Calgary, AB

Not Your Father's Resource Template

A C++ template for handling resources in a type-safe way

To C or Not to C...

It's very hard to find a real C compiler these days. All the ones I know of are really C++ compilers operating in backwards-compatibility mode. So I find it surprising that so much code is still being written in C rather than C++. Since C++ is in every way a superset of C there seems to be no reason for not taking advantage of at least some of the features of C++ that make the professional programmer's life so much easier — even just the simple ability to declare variables at their point of use rather than at the head of the current block or function.

Many programmers of course do use C++, although I see a lot of Macintosh code that still looks a lot like plain C — ranging from simple things such as using preprocessor macros instead of `const` literals or inline functions, to using global or static variables instead of a fully-encapsulated solution using objects.

Why should this be so? As with many things, there are probably a number of reasons. For example:

- Most programmers find it hard to justify the time to learn something new when they are functioning very productively with the skills they already have.
- Many of the standard C++ library functions do not fit well with the Mac OS APIs — by using C strings and pathnames for files, for example — and so it is easier just to stay with the C-style of programming adopted by Apple's Universal Headers.
- C++ is considered to be synonymous with object-oriented programming, and many people feel that the run-time overheads associated with this style of programming are too high in their situation.
- It is thought that using C++ on Mac OS requires the use of an application framework such as MacApp or PowerPlant, and since the program under development is not an application then C++ is not appropriate.

I plan to show how these and other reservations about C++ are actually misconceptions. Not everyone believes all of these things, of course, but for those still with doubts, or those who would welcome the chance to discover more about some of the power-features of C++ applied in a Mac OS context, read on.

Be Careful With That Chainsaw...

Someone has said that compared with other languages, C is like juggling with knives. Someone else has said that compared with C, C++ is like juggling with chainsaws — awesome power, but rather nasty when mishandled. Actually, I think that C++ has the capacity to be much safer than C. The increased ability to hide or localize dangerous operations in a well-tested submodule frees the programmer to concentrate on the logic of the application rather than avoiding all the usual "gotchas". What's more, courtesy of inline functions and stack-based objects this ability can come at zero runtime cost — as you will see.

Neil Mayhew works for Wycliffe Bible Translators, a non-profit organization dedicated to translating the Bible for the world's 400 million people that do not have it in their own language. Neil started programming in C in 1983, and graduated to the Mac in 1989. When he's not at his Mac or trying to beat his kids at video games you might find him flying a stunt kite if it's windy, or throwing a boomerang if it's not. Write to him at Neil_Mayhew@Wycliffe.org.

Many people are content to leave this work in the hands of the providers of application frameworks and class libraries. However, the real gains come when you develop data abstractions that are specific to the code in hand. This in turn requires an understanding of features such as template classes and smart pointers, which many people have not yet fully grasped and made their own. I hope that by the end of this series of articles you will feel that you have.

For those new to C++, I will now define some of the C++ jargon that is used throughout this article, but experienced readers may want to skip the rest of this section.

Data abstraction is the technique of creating an 'idealized' interface to a body of data that is independent of its actual representation. As far as possible, the abstraction should present the data as a single, easy-to-describe concept. For example, the abstraction of a stack could be an object with `push` and `pop` operations, rather than an array and a `next` pointer.

Overloading is the practice of assigning multiple meanings to the same identifier or symbol. In C++ this usually means defining several different functions with the same name but different sets of argument types, and allowing the compiler to distinguish which one should be called according to the types of arguments supplied. It can also mean that new versions of built-in operators (like +) are being defined, that take user-defined types as arguments. Note that overloading is not usually the same as overriding, since the latter refers exclusively to the redefinition of a superclass method in a subclass. A single class may overload several versions of one of its own method names, and overloaded operators are quite often not methods at all but global functions.

An *inline function* is defined just like any other function, and obeys all the same rules, but (usually at the compiler's discretion) its code is generated wherever the function is called, rather than just once where it is defined. In this way it is rather like a macro, except that all the rules for overloading, argument type-conversion and so on apply. It also avoids the typical macro problem of multiple-evaluation of arguments, just as with a 'real' function call. What's more, the compiler is free to optimize the generated code in any way that is consistent with the semantics of the function call, which can often bring huge gains in efficiency over a physical function call. Inline functions can reduce the cost of data abstraction to zero — yielding solutions that are as fast and compact as their unabstracted counterpart.

A *stack-based* object is one that is designed to be allocated just like any other local variable, rather than by a heap allocation via `new`. The space-allocation overhead is effectively zero (both in time and heap space) and the only execution overhead is in the constructor- and destructor-calls for the object. Very often, these calls represent code that would need to be executed anyway, for example to initialize a data structure or free some memory, and the compiler takes care of calling this code at its proper time.

A *template class* is a generic definition of an infinite family of related classes, in which each family member is associated with a different auxiliary data type. This association is made by substituting the specific data type for a formal argument of the template definition. For example, a stack template would define the behavior

Fast Mac ISAM Access

Puzzled by database performance issues?

Nothing beats... ISAM performance

Real-world data management solutions are typically more complex when one examines the pieces, than initially recognized by the majority of database programmers. All software projects are complex puzzles comprised of many details, most of which are data-related. Often today's "DBMS" solutions sacrifice the speed or control essential for a competitive application.

c-tree Plus®, by FairCom, has been the choice of commercial developers for twenty years precisely because it offers the flexibility and control at the detail level to fit a wide variety of data management needs. Proven on large Unix servers and workstations, c-tree Plus's small footprint and exceptional performance have also made it the engine of choice for professional developers on Mac and Windows. c-tree Plus offers sophisticated ISAM level control with which the developer may define precise data management solutions, making it a perfect fit for any development project requiring specific data handling features.

c-tree Plus® offers the most mature ISAM solution today...

FairCom's c-tree Plus database engine:

- Advanced Indexing Technology
- Complete Source Code
- Complete Transaction Processing
- ODBC Interface from Windows clients
- Over 25 Developer's Servers Included
- Royalty Free
- Standalone, Multi-user or Client/Server Models
- Y2K Compliant
- Supports Metrowerks, Symantec Compilers

The FairCom Server:

A solid, high performance database server that is scalable, portable and offers unequalled control. FairCom has been providing database solutions to the commercial development community for twenty years and supporting Mac for nearly as long. You won't find a better Mac DBMS solution, with these features and performance anywhere else!

- Client Side Source Code
- File Encryption
- File Mirroring Logic
- Full Conditional Index Support
- Full Heterogeneous Networking
- Multiple Communication Protocols: ADSP; SPX; TCP/IP
- Online Data Backup
- Small Memory Footprint
- Flexible OEM Licensing Options
- Source Code Availability

All these platforms supported in one package:

Mac, MIPS ABI, DEC Alpha, Sun SPARC, Windows 9X, SCO, 88OPEN, AIX, RS/6000, HP9000, Sun OS, Interactive Unix, Linux (Alpha...), AT&T System V, QNX, Free BSD, OS/2, Windows NT, Windows 3.1, DOS, Netware NLM, & Banyan VINES.

Don't wait, see for yourself! www.faircom.com
USA. 800.234.8180 Email: info@faircom.com

FairCom®

Database SOLUTIONS Since 1979

Phone: USA 573.445.6833 • EUROPE +39.035.773.464
JAPAN +81.59.229.7504 • BRAZIL +55.11.3872.9802

Other company, product and operating platform names are registered trademarks or trademarks of their respective owners.

of a family of stack classes, and each family member (called an *instantiation* of the template) would be a stack class storing some particular type (char, double, Person*, etc.). Template arguments are specified between wedges <>, and subsequently within the template definition they can be used just like a typedef name. The methods of a template class are usually inline, although they don't have to be.

A *smart pointer* is an object that has the syntax and semantics of a pointer, through overloading the * and -> operators for that class. Hopefully it will also have behavior that is useful enough to be considered smart! For example, a smart pointer class can be defined that automatically maintains a reference count on the objects being pointed to. Smart pointers are almost always stack-based objects, or data members of some other object. They are designed to be passed around by value rather than by reference (otherwise the reference counting would not work, for example). A native C++ pointer is often contrastingly referred to as a *raw pointer*.

Cast Your Cares Away...

Casts are a fact of life in Mac OS programming. Although they are messy and error-prone, they can't be avoided. For example, GetResource returns a Handle, but this must be cast to whatever structure represents the data before using it. However, a mismatch between the actual contents of the resource and the cast that is used is usually disastrous. The sophisticated type-matching features of C/C++ are designed to help you avoid this kind of thing, but a cast circumvents these checks completely. Whilst good use of C++ cannot remove the need for a cast somewhere in your code, it can at least confine it to one carefully-chosen place. The result, as you will see, is a much clearer and neater program, as well as a safer one.

As an example of using resources, we will consider a code fragment that reads and writes high-scores from a preference file (very loosely based on July's *Getting Started* column). The first thing is to define a structure that represents the layout of the data in the resource:

Listing 1.1: Resource structure

Score

```
struct Score
{
  UInt32   score;      // What the player scored
  Boolean  used;       // Whether this entry is in use
  Str255   name;       // The player's name (variable-length)
};
```

The code to read and display the high-score list might look as follows:

Listing 1.2: Traditional approach

ShowScores

```
const ResType kScoreType = 'Scor';

void ShowScores()
{
  int n = Count1Resources(kScoreType);

  for (int i = 1; i <= n; ++i)
  {
    Score** s = (Score**)Get1IndResource(kScoreType, i);
    if (!s)
      break;

    if ((**s).used)
      // Append (**s).score and (**s).name to score window...

    ReleaseResource((Handle)s);
  }
}
```

Good practice has been followed in using a symbolic definition for the resource type (although I have seen plenty of code that does not). However, note that:

- two casts are used (ugly and error-prone)
- there is no way to ensure that the cast and the ResType match (in the call to Get1IndResource)
- the resource-type symbol is used in two different places (an opportunity for mistakes if the name is ever changed)
- the cumbersome (**). syntax is needed for accessing the data.

How could we make use of C++ to overcome these deficiencies? Bear with me for a moment and take a look at one possible end result:

Listing 1.3: Alternative approach

ShowScores

```
typedef ResHandle<Score, 'Scor'> ScoreResource;

void ShowScores()
{
  ScoreResource s;

  int n = s.Count1();

  for (int i = 1; i <= n; ++i)
  {
    s.Get1Ind(i);

    if (!s)
      break;

    if (s->used)
      // Append s->score and s->name to score window...

    s.Release();
  }
}
```

The symbolic definition of the ResType constant has been replaced with a typedef representing the instantiation of a template (I'll explain this more fully in a minute), and the Resource Manager API's have been replaced with methods. In contrast with the previous list of deficiencies:

- no casts are visible
- there is no possibility of mismatch between ResType and pointer type, as the relevant information is locked together in the typedef
- the ResType does not need to appear anywhere except in the typedef
- the -> operator is used in place of (**).

Now a word about the typedef. A template class called ResHandle is defined in a header file, and is instantiated

(parameterized) with two pieces of information. One is the type of the data that the handle will refer to, and the other is the ResType constant that is associated with the data. Not many people realize that template parameters can be constant values as well as type names, but this is a very powerful feature of C++ that we will return to later. Finally, this instantiation of ResHandle is given the symbolic name ScoreResource.

You may also have realized by now that ResHandle is in fact a 'smart pointer'. It overloads the * and -> operators to perform the double-dereference for you (just as MPW C++ used to, way back). Some people would no doubt dismiss this as 'syntactic sugar' but I am in favor of anything that makes the code clearer and simpler. Of course, in one sense it makes the code more obscure, because it hides the fact that a handle is involved at all, but I think this is a price worth paying, especially if the ResHandle template is used universally throughout an application. Note that both . and -> are used on the ScoreResource object: the former calls methods that affect or use the value of the handle, and the latter allows one to access the object that is pointed to by the handle. This duality is very common with smart pointers, and it is important to understand the two different meanings clearly.

Under The Hood...

We can now take a look at the 'wizardry' behind the ResHandle template, although I hope you'll agree that in good C++, like Math, everything looks very simple once you have chosen the right definitions to use.

The original designers of the Mac OS came from an object-oriented background, and this is reflected in the fact that many of the Mac OS APIs are object-oriented in concept if not in syntax. For example, a resource handle "is a kind of" handle. All the calls that can be performed on a handle can be performed on a resource handle, and resource handles add a few extra calls of their own that can't be performed on a regular handle. So it would be nice if any C++ treatment of resource handles could reflect this inheritance relationship. This would have the added benefit of allowing compile-time detection of passing the wrong type of handle to an API — vastly preferable to discovering it at run-time, or even not discovering it at all.

So, the definition of ResHandle actually inherits from a more basic definition of MemHandle. We'll take a look at that first (**Listing 2.1**) and then proceed to ResHandle after that (**Listing 2.2**).

Listing 2.1: MemHandle template class

MemHandle

A smart-pointer template that encapsulates a regular Mac OS memory handle.

```
template<class Type>
class MemHandle
{
  protected:
    Handle h;

    // Basic casts - up/down refers to the 'class hierarchy'
    static Handle upCast(Type** t)
    {
      return reinterpret_cast<Handle>(t);
    }
    static Type** downCast(Handle h)
    {
      return reinterpret_cast<Type**>(h);
    }

  public:
    // Constructor - initialize the handle to zero
    MemHandle() : h(0) {}

    // Using compiler's copy constructor and assignment operator

    // Using compiler's destructor

    // Dereferencing operators
    Type* operator -> () { return *downCast(h); }
    Type& operator * () { return **downCast(h); }

    // Conversion operators
    operator Handle()    { return  h; }
    operator Type**()    { return  downCast(h); }
    operator Type* ()    { return *downCast(h); }

    // Status operators
    bool operator ! ()   { return h == 0; }
    operator bool  ()    { return h != 0; }

    // Mac OS APIs
    bool Allocate(Size n = sizeof(Type))
    {
      h = NewHandle(n);
      return MemError() == noErr;
    }
    bool Resize(Size n)
    {
      SetHandleSize(h, n);
      return MemError() == noErr;
    }
    bool Dispose()
    {
      DisposeHandle(h);
      h = 0;
      return MemError() == noErr;
    }
    // etc. etc.
```


http://www.scientific.com

Professional Software Developers

Looking for career opportunities?
Check out our website!

Nationwide Service
Employment Assistance
Resume Help
Marketability Assessment
Never a fee

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)
das@scientific.com

```
    OSErr Error()
    {
      return MemError();
    }
};
```

MemHandle explained

This template contains all the basic machinery needed to manipulate handles, although for simplicity of illustration quite a few methods have been left out (download the source code to see a slightly fuller implementation). It is defined as a template so that it can be used to represent any type of data that is held in a handle, acting much like a raw pointer. The compiler is then able to enforce proper type consistency throughout the program.

The template parameter, Type, is the data type of the contents of the handle. In our example, this is a Score. When used as a pointer, with -> or *, a MemHandle has the syntax and semantics of a Type*.

As far as data is concerned, MemHandle is just a wrapper for a raw Mac OS memory handle. A MemHandle, when allocated on the stack, takes up just the same amount of space. The data member, h, is declared protected so that subclasses (such as ResHandle) can easily access the raw handle. It could have been defined as a Type**, but the API-methods (of which there would be many in a full implementation) are greatly simplified if it is just a Handle.

All the casting is performed by two static, inline methods, upCast and downCast. The naming convention is based on the idea that a cast from a subclass to a superclass goes up the class hierarchy, and vice versa. Although a Handle is not actually a superclass of a Type**, the concept is the same. Internally, these two methods use reinterpret_cast rather than the traditional C cast. This is now preferred in C++, as a way to distinguish the code's intention from other uses of casting (such as static_cast, dynamic_cast and const_cast). By wrapping the casting inside these methods, all the other methods are made considerably cleaner and easier to read.

MemHandle defines a single constructor with no arguments, and allows the compiler to supply a standard copy-constructor (used for making a new MemHandle from another) and assignment operator (used for overwriting one MemHandle with another). Since the only legitmate way to create a non-nil Handle is by calling NewHandle or by copying another one, it makes sense to have these as the only constructors for MemHandle. If you want to allow copying of raw handles into smart ones, it's probably best to define a method (or explicit constructor) that takes a Type**, to avoid accidents. It's not wise to define a destructor that calls DisposeHandle because in the absence of reference-counting there is no way to be sure that another copy of the handle isn't still in use somewhere.

Whether or not you define them yourself, you should always give thought to "the fundamental four" whenever you design a new class: *default constructor* (no arguments, or arguments with default values), *copy constructor*, *assignment operator* and *destructor*. It's also a good idea to consider how you would define a less-than operator, especially if you ever want to store your object in an STL container. If the copy constructor or the assignment operator does not make sense for your object, then make it inaccessible by declaring it private and never implementing it (to stop the compiler implementing it for you). That way you get an error either at compile-time or at link time, but never at run-time.

The dereferencing operators are just simple calls to one of the casting functions, although the existence of these methods is the key to making our template be a smart pointer. The purpose of each is fairly self-evident, and the reason for the different return types is purely conventional.

Conversion operators allow one to supply a MemHandle whenever one of these other types is expected (such as in a parameter to a function or API), and the compiler will call the appropriate conversion code. Note, though, that because all the methods are inline, everything is resolved at compile-time and in the case of type-conversion no additional code will actually be generated.

The status operators simply provide a convenient way of checking whether the handle is currently nil or not. Statements such as if (mh) ... and if (!mh) ... will work as expected. Note that this is *not* the same as saying if (mh == 0) ... or if (mh != 0) ...

The rest of the class is API methods, and I have included just a few of these for illustration. Note that there needs to be a way to specify a size using Allocate and Resize since Type may actually have variable length (as in our Score example).

Listing 2.2: ResHandle template class

ResHandle

A smart-pointer template that encapsulates a Mac OS resource handle.

```
template<class Type, ResType kResType>
class ResHandle : public MemHandle<Type>
{
  public:
    // Constructors, etc. — using defaults

    // Mac OS APIs
    bool Get(int id, ResType type = kResType)
    {
      h = GetResource(type, id);
      return ResError() == noErr;
    }
    bool Get1(int id, ResType type = kResType)
    {
      h = Get1Resource(type, id);
      return ResError() == noErr;
    }
    bool GetInd(int i, ResType type = kResType)
    {
      h = GetIndResource(type, i);
      return ResError() == noErr;
    }
    bool Get1Ind(int i, ResType type = kResType)
    {
      h = Get1IndResource(type, i);
      return ResError() == noErr;
    }

    static short Count(ResType type = kResType)
    {
      return CountResources(type);
    }
    static short Count1(ResType type = kResType)
    {
      return Count1Resources(type);
    }
```

```
    void Changed()
    {
      ChangedResource(h);
    }
    void Release()
    {
      ReleaseResource(h);
      h = 0;
    }
    // etc. etc.

    static OSErr Error()
    {
      return ResError();
    }
};
```

ResHandle explained

Apart from one subtlety in the template parameters, ResHandle just adds some extra API methods. All the basic functionality, and the tricky operator overloading, is inherited from MemHandle.

ResHandle has two template arguments. The first is simply passed on down to MemHandle, and is used to define the data type of the contents of the handle. The second, however, is not allowed to be a class or type name at all. It has to be a value of type ResType, and a constant one at that. As we have seen from the Score example, this would normally be a literal four-char code supplied as part of a typedef:

```
typedef ResHandle<Score, 'Scor'> ScoreResource;
```

So this constant value actually becomes a part of the datatype defined by the typedef. This may seem a little hard to grasp at first, but a couple of other examples may make things clearer. In traditional C, it is possible to write:

```
typedef char Acronym[4];
Acronym* myList;
```

Then the compiler knows that Acronym means "array of four characters," and when we write myList++ the compiler knows to increment the binary value by 4. So the constant value 4 becomes a part of the type definition.

C++ has extended this concept, mostly to allow generic implementation of fixed-size arrays using a template (useful for Pascal strings). So in C++ we could write:

```
template<int nchars> VariableAcronym
{
  char data[nchars];
  ...
};

typedef VariableAcronym<4> Acronym;
```

Of course, the constant-template-parameter mechanism can be used for any purpose we like, and in this case it comes in extremely handy as a way of recording the ResType value that belongs with a particular type of resource handle. By making it a template parameter, we never have to store it at runtime along with the handle, because everywhere it is needed we can supply it as a compile-time constant (kResType). Of course, the value of kResType will be different for each instantiation of the template.

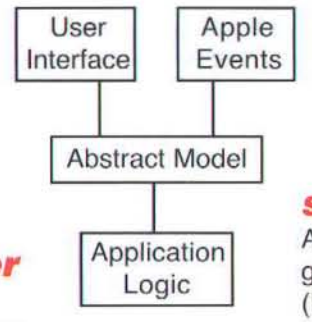
A different value for the resource type can however be provided at runtime by supplying an optional second argument to

Fight Boredom

Let's face it: Much of programming is boring and repetitive. Well, that's where the right tool can save days, weeks, or even months of your valuable time.

AppMaker
Your Assistant Programmer

AppMaker makes it faster and easier to make an application. It's like having your own assistant programmer. You point and click to tell AppMaker the results you want, then it generates "human, professional quality code" to implement your design.



Model-View-Controller

AppMaker's generated code uses the MVC paradigm. It separates the user interface from application logic, making code easier to write. You deal only with abstract data; AppMaker takes care of the user interface.

Scriptable Applications

AppMaker generates the 'aete' resource and generates code to access your data (Properties and Elements in the Apple Event Object Model) and to handle Events.

Just $199 from www.devdepot.com

B•O•W•E•R•S Development

P.O. Box 929, Grantham, NH 03753 • (603) 863-0945 • FAX 863-3857
bowersdev@aol.com • http://members.aol.com/bowersdev

MacTech CD-ROM

Get the info you need in one shot!

The *MacTech* CD-ROM with THINK Reference is the essential reference resource for Macintosh programmers. This CD includes the THINK Reference personal database system and a wealth of Macintosh programming databases, featuring almost 170 issues of the journal of Macintosh programming – *MacTech Magazine*. This release also features the THINK Reference Compiler, which allows you to compile HTML files into your own compact, searchable THINK Reference databases, and the THINK Reference Collector for building new Mac OS API Databases.

MacTech CD-ROM
with THINK Reference™

PO Box 5200 • Westlake Village, CA • 91359-5200
800/MACDEV-1 (800/622-3381) • Outside US/Canada: 805/494-9797
Fax: 805/494-9798 • E-mail: orders@devdepot.com

the Get methods. This is to allow for resource types that sometimes masquerade as a different ResType (owner resources, for example, which are really 'STR 's). To get the owner resource, you would declare a variable of type StrResource and pass the application's signature to the Get call.

The Count methods have been defined static, because they don't act on any particular handle. They do, however, fill in kResType for us, which is handy. In the Score example that we looked at earlier, the declaration of the ScoreResource handle was moved up out of the loop so that we could use it to count the number of score resources. We could, of course, call ScoreResource::Count1 directly, but that would be more verbose and more error-prone as the name ScoreResource would have to appear twice, once in the method call and once in the handle declaration.

The Final Score...

So how does our solution stack up against the list of problems given at the start of this article?

- *Taking time to learn something new:* I hope you'll agree that the techniques I have outlined do bring real benefits to our code. I hope too you'll have found that learning something new didn't take up all that much of your valuable time.
- *Living with the Mac OS APIs:* the C++ that we wrote gets along just fine with the APIs, and makes them a lot easier to work with into the bargain.
- *Run-time overheads:* using our ResHandle template uses no more memory, generates no more code, and executes just as quickly as our traditional solution. Actually, the constructor does redundantly initialize the handle to zero once, at the start of the function, but the compiler's optimizations remove this (as shown by a disassembly).
- *Application frameworks:* we relied on nothing more than the Universal Headers, and yet managed to write some fairly effective C++. The low run-time overhead means that this code could be used without problem in a code resource or system extension (an entire example application is 2200 bytes code and 1800 bytes data when built for PPC). Equally, this code would fit quite comfortably into any framework-based application too.

Incidentally, PowerPlant and MacApp both have suites of utility classes that can be used independently of the rest of the framework, and you should check these out before rushing ahead to reinvent too many more wheels. However, it is hard to find a generic solution that fits everyone's needs, so there is still great benefit in being able to develop your own utility classes, or customize those written by others.

The code presented here has necessarily been rather short and simple, of course. It is intended only as an illustration of the kinds of things that become possible when the full power of C++ is brought to bear on the task of programming for the Mac OS. In a future article, I'll present a lightweight ostream-style class that is based around Pascal strings. As well as being invaluable for generating text messages that need to appear in the UI, it provides further illustration of the advanced capabilities of C++. MT

By Kas Thomas

Dynamic PDF Made Easy

It's simpler than you think to generate custom PDF documents in real time using Perl and CGI

Adobe's Portable Document Format (PDF) has won universal acclaim as document interchange format, because of its ability to deliver graphically rich, structured content consistently across multiple operating environments. Because of its strong support for vector graphics, font embedding, hypertext links, and other advanced features, PDF is a powerful, far-reaching document standard. But that also makes it a relatively *complex* standard. (For details, see the September 1999 *MacTech*.) PDF's complexity, in turn, has discouraged many web developers from attempting to write PDF files dynamically, at the server. Automatically generated HTML pages are common; but when was the last time you saw an auto-generated PDF web page?

It turns out that with a little foreknowledge of PDF's internal workings, and a good grasp of CGI fundamentals, it's not that hard to put together server scripts that will generate PDF on demand. There's no question that dynamic PDF can be a challenge to do right. But if your dynamic document needs are modest (for example, if you'd simply like to be able to generate a custom "Thank You" page at run time) and your Perl skills are passable (i.e., you've graduated from writing "Hello World" to generating the much more impressive string "Internal Server Error"), you can automate the serving of custom PDF pages with surprisingly little effort. In fact, a Perl script comprising no more than five or six dozen lines of code will get you started.

Our Strategy

Our strategy will be simple: We want to be able to collect an arbitrary glob of "user text" from a web form, and convert that text to a PDF page served back to the user via HTTP, suitable for viewing in a browser equipped with the free Acrobat Viewer plug-in. For simplicity's sake, we'll start by omitting any attempts at vector graphics, colored text, rotated text, styled text in various point sizes, etc. (But I hasten to add that before this article is finished, we're going to be doing all of those things and more.) For starters, we just want to serve the user's words back to him, at any convenient point size, in black and white, on a letter-sized PDF page. For sake of convenience, we're also going to generate the "data collection" form and our PDF reply from one and the same Perl script. That way, we don't have to keep track of multiple documents: an HTML form, Perl scripts, CSS stylesheets, etc. If you want to add that complexity to the picture yourself later, so be it. Here, we're going to keep things stone-simple.

I might point out that "live" copies of the dynamic-PDF scripts developed below can be found at <http://www.acroforms.com/dynamicPDF.html> (which is a portal page with links to several working scripts).

I also want to stress that when we're done, the PDF pages we will have generated can be saved to disk using your browser's Save As feature. (Reader won't save the docs, but your browser will.) Thus, we will have achieved the Holy Grail of the PDF world: writing PDF without using Distiller or Acrobat.

Once we've gotten our basic approach nailed down, we'll branch out into fancy features like colored backgrounds, stroked/filled paths, rotated text, styled text, etc. But first, a detour into the land of PDF internal structure.

Kas Thomas <kt@svgcentral.com> consults for Adobe Systems and writes a regular column about PDF-related programming (covering JavaScript, CGI, dynamic PDF, and related issues) for www.acroforms.com. He has been programming in C and assembly on the Mac since MC68000 days and remembers when the MacOS ran in black-and-white.

PDF at the Subatomic Level

Internally, a PDF file consists of plain-ASCII object descriptions, following a notably Postscript-like (i.e., postfix-notation) syntax. All PDF files have a header (containing version information), a cross-reference (or 'xref') table containing offsets to the various objects that make up the file, and a trailer containing references to the root object as well as the byte offset from the start of the file to the beginning of the 'xref' table. (See my story in see the September 1999 *MacTech* for more information.) The formal PDF specification, available online at http://partners.adobe.com/asn/developer/PDFS/TN/PDFSPEC.PDF, describes the relationships between PDF objects (and the meanings of various dictionary "key" entries) in comprehensive detail, if you're interested. But we're going to skip all that, because most of it is not necessary for what we're doing.

It turns out you can bend (or even break) a lot of the "rules" spelled out in the PDF spec, without making Acrobat Reader unhappy, if you know a thing or two about PDF internals. An instructive exercise in this regard is to make Adobe's Distiller program generate a small "Hello World" type of file, then examine the file in a text editor—and try to find superfluous data in it that can be removed. If you try this, you will quickly discover that /Info objects, for example, contain meta-data about the PDF file (such as the doc's author, subject, producer, keywords, etc.) that can easily be dispensed with. Likewise, Adobe tends to insert long /ID objects (containing a kind of "digital fingerprint" for identification of the file) in PDF files; these aren't strictly needed.

If you keep removing superfluous objects and data items from a PDF file, you may be amazed at how much "data" you can actually get rid of without making Reader complain. A careful re-reading of the PDF Specification usually reveals that items you *thought* were mandatory are actually optional. Even the all-important 'xref' table isn't strictly needed, since Acrobat Reader can (and will, and does) generate the necessary byte offsets itself, at runtime, if the table is defective or missing.

In late 1999, I sponsored a contest at <www.acroforms.com> in which the goal was to produce the smallest possible PDF file that wrote "Hello World" to the screen without making Acrobat Reader generate any error dialogs. Amazingly, the winner of that contest submitted a file that was only a little more than 200 bytes long:

```
%PDF-1.
1 0 obj<</Pages 2 0 R>>
2 0 obj<</Kids[<</Parent 2 0
R/Resources<</Font<</R
<</Subtype/Type1/BaseFont/Arial>>
>>>>/Contents<<>>stream
BT/R 14
Tf(Hello World!)Tj
endstream
>>]>>
trailer<</Root 1 0 R>>
```

Note that the second object (starting with "2 0 obj...") is extra-long, with many nested "dictionary" entries, ending only just before the word "trailer" at the bottom of the file. This particular PDF file uses 14-point Arial as the typeface. No font encodings are embedded in the file, however, because Arial is one of the base-14 fonts that Acrobat Reader ships with. (Arial replaced Helvetica in the version-4.0 release of Reader.) Every machine that has Acrobat Reader is guaranteed to have Arial.

There are many things technically "wrong" with this short PDF file, including the fact that it has no 'xref' table and contains a text stream that is opened with 'BT' (for Begin Text) but is never closed with 'ET' (End Text). In many ways, it's a wonder Reader can even parse and display this file. Yet it does.

There is one small technical difficulty with this file that impacts viewing: Namely, no *positioning information* is given for the text — which means (since Reader "zeroes out" the current transformation matrix, or CTM, before displaying any block of text) that the starting "pen position" for displaying "Hello World!" in this instance is the origin of user space: i.e., the lower left corner of the page. You have to scroll down to the very bottom of the page to see the single line of text. To fix this requires that we add a transformation matrix of our own to the text stream:

```
%PDF-1.
1 0 obj<</Pages 2 0 R>>
2 0 obj<</Kids[<</Parent 2 0
R/Resources<</Font<</R
<</Subtype/Type1/BaseFont/Arial>>
>>>>/Contents<<>>stream
BT/R 14
1 0 0 1 72 720 Tm
Tf(Hello World!)Tj
endstream
>>]>>
trailer<</Root 1 0 R>>
```

Can you spot the change? We've added a line that ends with 'Tm' (the transform-matrix operator). The six-number matrix will look familiar to you if you've studied Postscript. The first four numbers are scaling and skewing factors (signifying no change in those characteristics, in this case); the final two numbers are translation values in 'x' and 'y', respectively. Since the default user space in PDF has 72 units to the inch, using values of 72 and 720 result in the starting position for our text results in our text being drawn one inch from the left edge of the page and ten inches up from the bottom. (In PDF, 'y' units get bigger as you go *up*.) Now our 14-point text will be drawn where the user can see it.

In the Perl script that follows, we're going to use a variation of the foregoing PDF file as the basis (the "template," if you will) for our dynamically generated PDF page. While some experts may cringe at the thought of using a "broken" PDF file as the starting point for this kind of exercise, the fact is that for illustrative purposes, a "subatomic" template file of this kind is hard to beat. Besides, the proof of the pudding is that Acrobat Reader doesn't in any way "choke" on the end result. What the user sees in his (or her) browser is a fully functional PDF file: one that can be saved to disk (using the browser's Save As feature) for later use.

The Common Gateway

Collecting information from users is a common task in the world of the Web, calling for the use of interactive forms. Sometimes, the form in question is a static HTML file stored on

a server; but increasingly, HTML forms themselves are dynamically generated at the server in response to user requests. That is, the form itself doesn't exist until a CGI script, often written in Perl, generates it.

CGI (the Common Gateway Interface) is nothing more than a set of conventions for pushing and pulling information across an HTTP connection. Forms, and the scripts that retrieve information from web forms, use the CGI protocol to get the job done. Learning to write CGI scripts in Perl is an art unto itself, but the job is made tremendously easier if you take advantage of some of the excellent free Perl libraries available for implementing CGI processes.

Arguably the best Perl library for this task is Lincoln Stein's powerful CGI.pm package. This set of routines (which takes the pain out of writing HTML at the server and dealing with web forms) is so useful that it now ships with Perl as part of the "full install" of the language. What this means is that any Unix server that has Perl 5.003, patchlevel 7 or higher (as almost all do) automatically has CGI.pm, waiting to be called from any Perl script.

When you use CGI.pm, you don't have to worry about whether incoming form data is being supplied via GET or POST, where it's being stored on the server, or any of the nasty parsing details involved in extracting urlencoded data from HTTP streams. To get the value of a form field called "UserName" from a Perl script at runtime using CGI.pm, all you have to do is:

```
$name = param('UserName');
```

Here, $name refers to the local scalar variable into which we wish to store the value from "UserName." If the user has typed "John Smith" into the relevant form field before Submitting the form data to the server, then $name will contain "John Smith" after this line of code executes. The param() function in CGI.pm takes care of finding the incoming data, decoding it if it's urlencoded, etc.

Dynamic Forms Using CGI.pm

One task CGI.pm excels at is generating HTML code dynamically. For example, to create the HTML form shown in **Figure 1**, all we have to do is execute the following few lines of Perl:

```
print header,
  start_html( title=>'PDF Bounceback, by Kas Thomas',
              bgcolor=>'#FFFFFF'),
  p(h2('PDF Text Entry'));

print start_form,
  textarea(-name=>'UsersText',
           -rows=>24,
           -columns=>60),p,
      submit( name=>'action',-value=>'Generate PDF');

end_form;
```

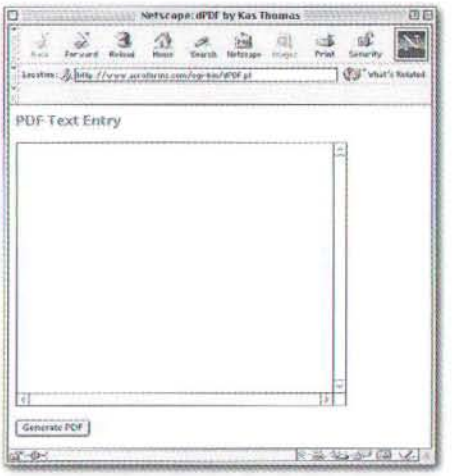


Figure. 1. *A form created dynamically using CGI.pm.*

The function h2('content here') applies an HTML level-2 headline tag around the text "content here"; likewise, p() applies paragraph tags around a block of text, and so forth. Nesting the function calls causes the relevant HTML tags to nest correctly. A common idiom in Perl for getting multiple strings to print to an output stream is to separate the strings with commas and put them between print at the start and a semicolon at the end. Functions like header(), start_form(), and end_form() are part of the CGI.pm package. (In Perl, parentheses after a function name

Macintosh File Sharing using Internet Protocols

ShareWay IP

Now available with the DoorStop Personal firewall for a special price!

NOW PART OF MAC OS 9!

When Apple wanted to add TCP/IP File Sharing to Mac OS 9, they chose Open Door's ShareWay™ IP. ShareWay IP makes Internet File Sharing just as easy as everything else on the Mac.

IF YOU RUN MAC OS 9, you can upgrade its IP File Sharing to ShareWay IP 3.0. ShareWay IP 3.0 adds logging, enhanced security and other features.

IF YOU DON'T RUN MAC OS 9, ShareWay IP 3.0 works on any Mac back to System 7.5.5 (even 68K's), giving those Macs the same IP File Sharing capabilities as Mac OS 9.

Open Door Networks, Inc.
800/480-DOOR
sales@opendoor.com

Open Door Networks

http://www.opendoor.com/swip/

do not constitute a "function call operator." They are therefore optional, if no arguments are given.)

If you write a Perl script containing the foregoing lines of code, name the script "Form.pl", and place it in the "cgi-bin" directory of your web server, then whenever anybody goes to the script's URL, the script will launch automatically and write the form shown in **Figure 1** to the caller's browser. This is how many (if not most) dynamic forms work.

A Dynamic PDF Script

At this point, believe it or not, we're in a position to put together a full, working Perl script for producing an HTML form dynamically, retrieving the contents of that (filled-out) form, and auto-generating a PDF reply back to the user's browser. **Listing 1** shows such a script, complete.

Listing 1: dPDF.pl

dPDF.pl

A complete Perl script for generating an HTML form, retrieving the form's contents, and writing the contents back out as a PDF file.

```
#!/usr/bin/perl
# ----------------------------------
# Script: dPDF.pl (simple script to output PDF dynamically)
# © 2000 by Kas Thomas. Updates at www.acroforms.com.
# ----------------------------------
use CGI qw/:standard/;

# Check params, and if null, show form...
DrawForm() unless param();

$allText = param('UsersText');

print header("application/pdf");    # output PDF header

Send_PDF_Leader();

foreach $line (split /\n/,$allText) {

   print '(' . $line . ')Tj';
     print "\n";
   print 'T*';          # goto start of next line
     print "\n";

}

Send_PDF_Trailer();

# - - - - - - - - - - - DrawForm() - - - - - - - - - - -
sub DrawForm {

  print header,
    start_html(-title=>'PDF Bounceback, by Kas Thomas',
                -bgcolor=>'#FFFFFF'),
    p(h2(font({-color=>'red'},'PDF Text Entry')));

  print start_form,
    textarea(-name=>'UsersText',
             -rows=>24,
             -columns=>60,
             -wrap=>virtual),p,
        submit(-name=>'action',-value=>'Generate PDF');

    end_form;

    exit;
}

# - - - - - - - - - - Send_PDF_Leader() - - - - - - - - -
sub Send_PDF_Leader {

print <<PDF_LEADER;
%PDF-1.
1 0 obj<</Pages 2 0 R>>
2 0 obj<</Kids[<</Parent 2 0 R/Resources<</Font<</R
```

Cookies in PDF?

With hyperlinks, styled text, form widgets, and JavaScript all finding their way into more and more PDF pages these days, it would appear that PDF pages are (in theory, at least) capable of emulating a surprising number of features found in HTML web pages. But what about storing cookies? Can PDF do that? Not technically. But a little-appreciated feature of PDF's JavaScript facility lets you put persistent data on a user's machine for later retrieval (unbeknownst to the user)—which is, of course, the basis for most of the behavior associated with "cookies."

PDF's JavaScript binding allows users of Acrobat (the full commercial product; not Reader) to attach scripts to button events, page-open actions, and various other "trigger points" in a PDF document. Among the built-in classes existing in Adobe's Acrobat extensions to core JavaScript is a Global object class. One of the methods of this class is called **setPersistent()**. Applying this method to a global variable makes the value stored in that variable persistent across Acrobat or Reader sessions. The way it works is that Acrobat or Reader will, upon encountering a variable that has been made persistent, write that variable out to disk on program shutdown. (Conversely, it reads such variables from disk on launch.) The persistent data are written to a file in the Plug-Ins hierarchy called **glob.js**.

What is little-appreciated is the fact that even Acrobat Reader (which is supposedly a read-only program) can write a **glob.js** file to disk—and will do so, if a file contains persistent data created in JavaScript. Let's say you want to store the string value "April 30, 2000" persistently for later retrieval (possibly from a different PDF file altogether). You could attach the following bit of JavaScript to a Page Open action in your PDF file:

```
// create a global property, myString, and give it a value:
global.myString = "April 30, 2000";

// make it persist:
global.setPersistent("myString");
```

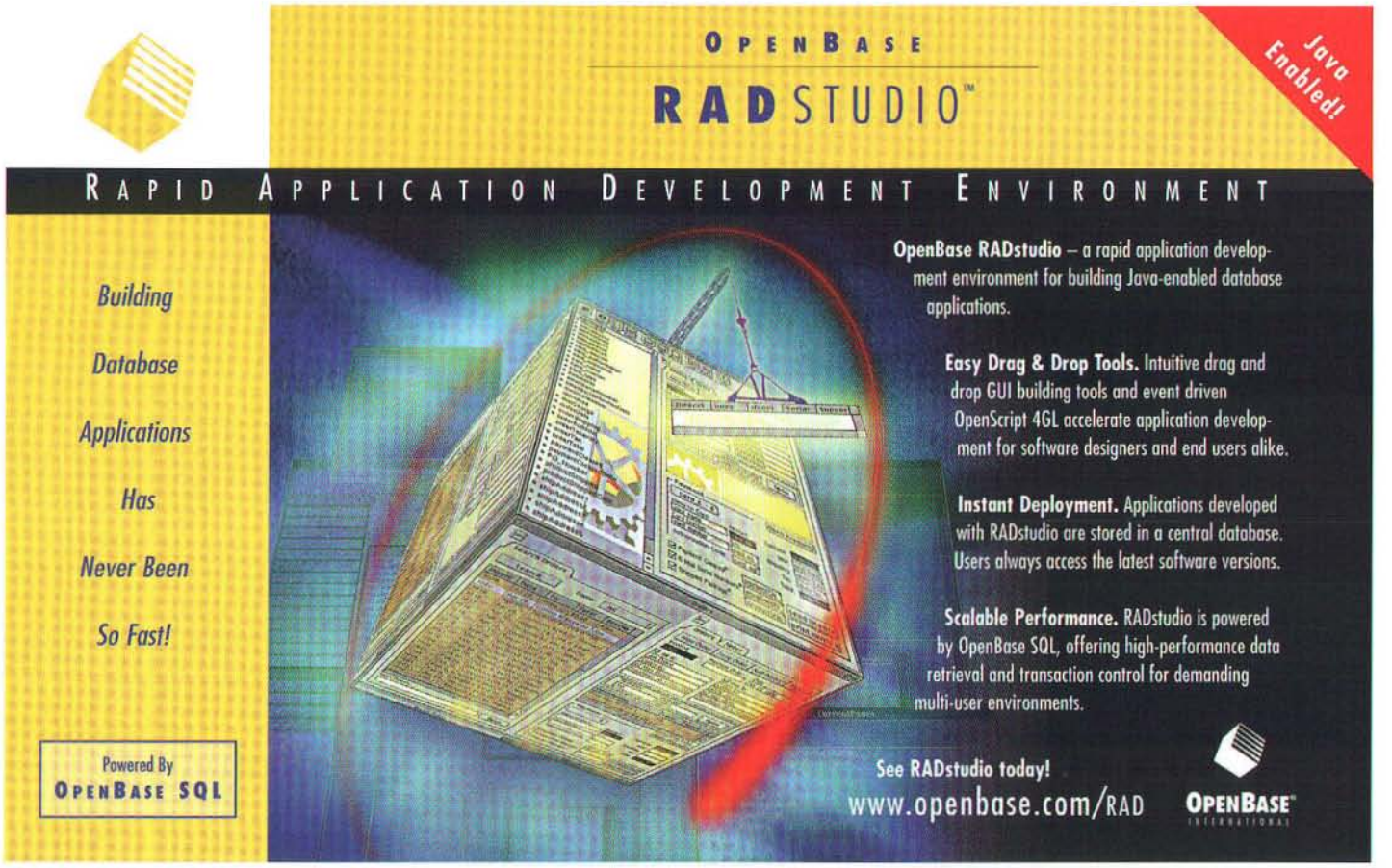
When these two lines execute, it tells Reader (or Acrobat) that the value in "myString" needs to be stored to disk, in the **glob.js** file, at program shutdown. Later, another PDF file can execute JavaScript containing the following:

```
// create a blank string:
var lastSessionData = new String();

// retrieve 'myString'
if (typeof (global.myString) != 'undefined')
lastSessionData = global.myString;
```

The (local) variable **lastSessionData** will now contain "April 30, 2000."

If you're ever curious as to what's in your **glob.js** file, just look for it on disk (with Sherlock) and open it in SimpleText. It's a plain ASCII file.

Yum.—*Kas Thomas*




```
<</Subtype/Type1/BaseFont/Arial>>
>>>>/Contents<<>>stream
BT/R 14
1 0 0 1 72 700 Tm
15 TL
Tf()Tj
PDF_LEADER
]

#  - - - - - - - - - - Send_PDF_Trailer() - - - - - - - - -
sub Send_PDF_Trailer {

print <<PDF_TRAILER;
endstream
>>]>>
trailer<</Root 1 0 R>>
PDF_TRAILER
}
```

After a use directive to make sure Perl knows we need to use the CGI.pm package, we check to see whether the CGI.pm function param() returns nothing, in which case it means the script is being called for the first time. If the script is being called for the first time, it means we need to generate the HTML form of **Figure 1** — which is accomplished by the subroutine DrawForm(). Otherwise, if param() is not nil, we're responding to a hit on the "Generate PDF" button (which is the form's equivalent of a Submit button).

If we're responding to a hit on the Generate PDF button, the first order of business is to retrieve the user-entered text, which we do with:

```
$allText = param('UsersText');
```

The textarea field of the HTML form is named "UsersText"; hence, we want to retrieve whatever the user typed there. The foregoing line of code captures the user's text into a single big string (a scalar variable, in Perl parlance) called $allText. We'll have occasion to parse that big string into smaller constituents (representing individual *lines* of text) in a moment.

Before we can write PDF to the user's browser, we have to open an HTTP connection and send an HTTP header with the appropriate MIME type for PDF (which is "application/pdf"). This is the browser's way of knowing what kind of file is about to arrive. The browser can take appropriate action in terms of launching helper apps, handing off control to plug-ins, or handling the transaction itself. In this instance, if the user has the Acrobat browser plug-in installed, the browser will hand control to the plug-in. It's also possible that the user has set the browser up such that incoming PDF files are redirected to the standalone Acrobat Reader app. If the MIME type "application/pdf" has *not* been registered with the browser, the browser will react to the incoming header by asking the user what to do with the about-to-arrive unknown file type. The user can then cancel out of the transaction, store the incoming file to disk, etc.

Opening the HTTP connection and sending the proper header is done with:

```
print header("application/pdf");
```

After that, our script goes to a subroutine called Send_PDF_Leader(). This subroutine simply writes 200 bytes or so of raw PDF to the HTTP connection. Most of that code should look familiar to you from our discussion of minimal PDF files earlier.

Writing the User's Text

Next comes our script's "main loop," you might say, in which we parse the user's text into individual lines and write them out as PDF text-stream data. We use Perl's split() function to divide the $allText string into substrings, splitting at every newline. Then we write a line of user text, in parentheses, followed by the 'Tj' operator (PDF's "writeline" operator), and after every line of text so processed, we write a 'T*', which (in PDF) means to drop down to the start of the next line (using the current leading) and prepare the virtual pen to write more text.

After writing every line of user text, we call Send_PDF_Trailer() to write out the last few bytes of our PDF template. At that point, we're finished.

An example of a dynamically generated PDF page using this script is shown in **Figure 2**.

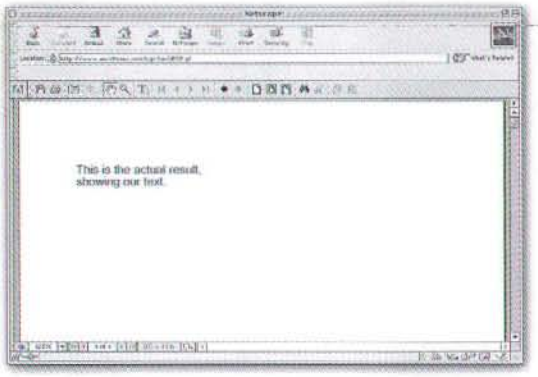


Figure. 2. *A dynamically generated PDF file.*

The Great Escape

At this point, we have a script that obediently writes our user's text input as 14-point Arial, in black ink on a white PDF page. Not very interesting-looking. It would be nice if we could jazz things up a bit by drawing our text in color, perhaps in various additional sizes. It would also be nice if we could specify something other than a pure white background. But before we proceed to increase the spiff factor of our PDF, we need to attend to one small detail.

PDF represents strings internally as ASCII character sequences inside parentheses. Thus, whenever a string contains parentheses as part of the text, they need to be "escaped" with a preceding backslash. Otherwise, the PDF interpreter, upon encountering a right-parenthesis inside a block of text, may think the text string has been terminated! The right-paren will be treated as an end-of-string marker. This could lead to strange errors if our user is in the habit (as many of us are) of loading up text with parenthetical remarks.

Fortunately, it's not hard to trap parens and "escape them" automatically for the user. Of course, we also have to trap and escape *backslashes* in the user's text, because (again) PDF treats all backslashes as "special" in a text stream. To deal with parens and backslashes, we declare an associative array in Perl as follows:

```
%PDF_ESCAPES = ('\\' => '\\\\',
                ')'  => '\)',
                '('  => '\(');
```

The way this special kind of array (sometimes called a *hash*) works is that when we want to look up the "replacement" for, say, a bare right-paren, we just need to do:

```
$PDF_ESCAPES{ ')' };
```

Note the dollar sign preceding the hash name (and the curly braces enclosing the array index). In Perl, the dollar sign means we are coercing the array lookup into a *scalar context*. When the above statement evaluates, it will evaluate to the lookup value '\)' (i.e., the string representing our properly escaped paren).

Of course, Perl (like PDF) *also* treats backslashes as special, which means that a string representing a literal backslash needs to be escaped — with a backslash! That is, the string '\\' prints as one backslash (not two) in Perl. Likewise, '\\\\' is treated as two literal backslashes in a row, not four. That's why the first lookup item in our hash, namely '\\\\', is associated with '\\'. (Confused yet?)

Now. We need to check every line of user input before writing it as raw PDF, so that any parens or backslashes contained in the user's text get converted to their properly escaped equivalents, preventing Acrobat Reader from choking. We can accomplish this essential check with the following line of code, inserted in our "main loop":

```
$line =~ s/([\\()])/$PDF_ESCAPES{$1}/ge;
```

"Wow," you're probably muttering, "what a weird-looking line of code!" And indeed it is. A Perl fan will recognize it immediately as a powerful *regex-based substitution* (regex being short for "regular expression"). In Perl, a statement like

```
$myString =~ s/this/that/g;
```

means to scan the contents of $myString, looking for the substring 'this', and upon finding a match, *substitute* (per the leading lowercase 's') the string 'that'; and oh, by the way, do this *repeatedly* (globally) throughout the entire contents of $myString. (That's what the trailing lowercase 'g' means.)

In our somewhat complicated substitution operation, we're scanning for any match of the regular expression [\\()], which is a *character class* with three members: the backslash character (which we have to escape with a backslash, for Perl's benefit; hence '\\'), the left-paren, and the right-paren. Square brackets mean that these characters form a class of allowable "match" characters. Any match of any one of these characters (in any order) inside the target string will trigger a "hit." Enclosing the entire character class *within parentheses*, as in ([\\()]), means we want Perl to store a *temporary copy* of any match that occurs, in a variable called $1. (If you're not familiar with this idiom, don't spend too much time worrying about it right now. You can always read up on regexes later.) Our replacement value, which Perl consults whenever a "hit" happens, in this case is $PDF_ESCAPES{$1} — that is to say, we use the match value ($1) as an index into our associative array, in order to retrieve the appropriate replacement string. This trick will only work, however, if we remember to put a trailing lowercase 'e' on the end of the expression, which means *evaluate the replacement string as a Perl expression*. Otherwise, Perl will literally substitute whatever's in the replacement position, unevaluated. That's not what we want.

Upping the Spiff Factor

As I started to say before, it would be nice if we could "spiff up" our PDF output a bit, augmenting it with colored text, styled text, or perhaps some vector graphics or different background colors, etc. — anything except plain old black text on a white background.

Adding extra spiff turns out to be astonishingly easy. All we need to do is let the user "drop down" into low-level PDF whenever he or she wants to. Follow me for a minute. *Suppose* we invent a special "escape command" of our own, signified by (let's say) the ASCII string <@>. We'll call this the "escape-to-PDF" operator. To get back to normal text-writing mode, we'll require the use of </@>, which we'll call the "escape-to-text" operator. Anything bracketed by <@> and </@> gets treated as raw PDF. While your user is in "raw PDF" mode, he can make use of any of the Postscript-like page operators and vector-drawing commands recognized by the PDF Specification: **m** for moveto, **l** for lineto, **rg** for setting the RGB color, **Tm** for set matrix, **B** for fill and stroke path, etc. (See the URL given earlier for the formal PDF Specification, at Adobe's web site.)

Using the same regex-based substitution trick that we used before, we can add one more lookup table (or associative array) to our script, and add one more regex-substitution line to our main loop. Only this time, the hash will return a value of ')Tj ' for any hit on <@>, and a value of ' (' for any hit on </@>. That way, whenever the user types <@> as part of his text stream, our script outputs a PDF end-of-string marker, followed by the PDF "write line" operator (Tj), spoofing Acrobat Reader into letting the user write raw page operators, if he wants to. When the user types </@>, the script outputs a "begin string" marker, and Reader is spoofed into going back into text-stream mode.

What Can You Do With It?

Now we've got quite a bit of power in our hands. If you know a little bit of PDF, you can create almost any kind of display magic you want with your dynamically generated PDF document. Let's try the following code entered into our form's main text area:

```
<@>0.85 0.85 0.85 RG</@>
<@>1 J 110 w 250 600 m 250 600 l B</@>
<@>.6 .6 .6 RG</@>
<@>18 w 72 72 m 72 720 l 540 720 l 540 72 l s</@>
<@>0 1 -1 0 144 520 Tm</@>
Let's try some rotated text.
```

The end result is shown in **Figure 3**. We've created a PDF document containing a large grey box (going around the edges of the page), a filed grey circle, and the words "Let's try some rotated text" running vertically up the page.

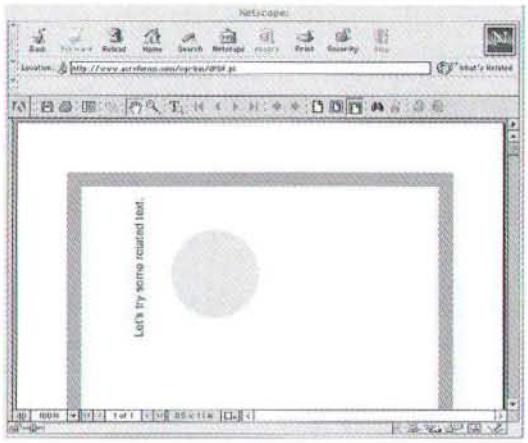
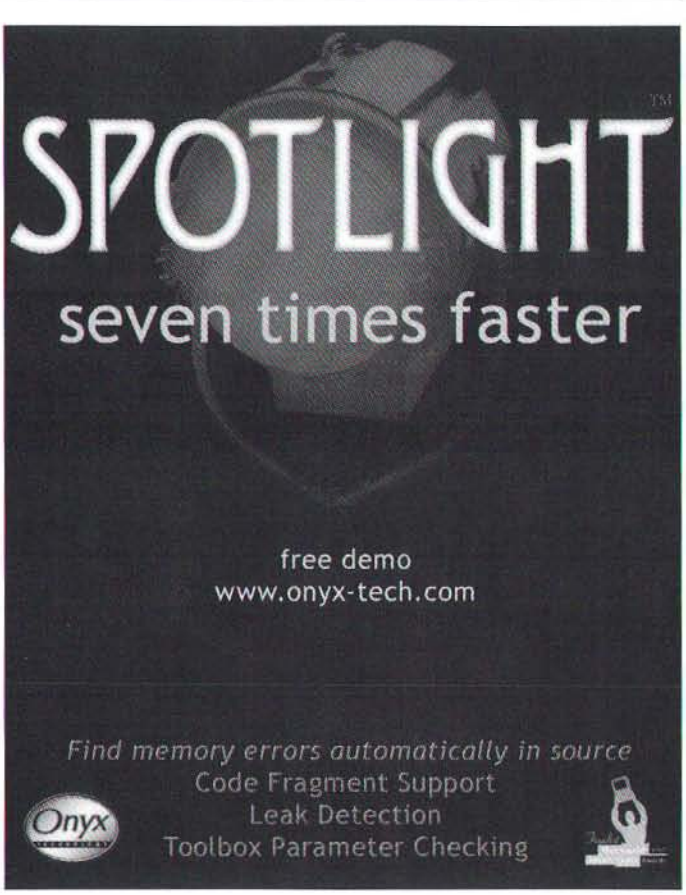


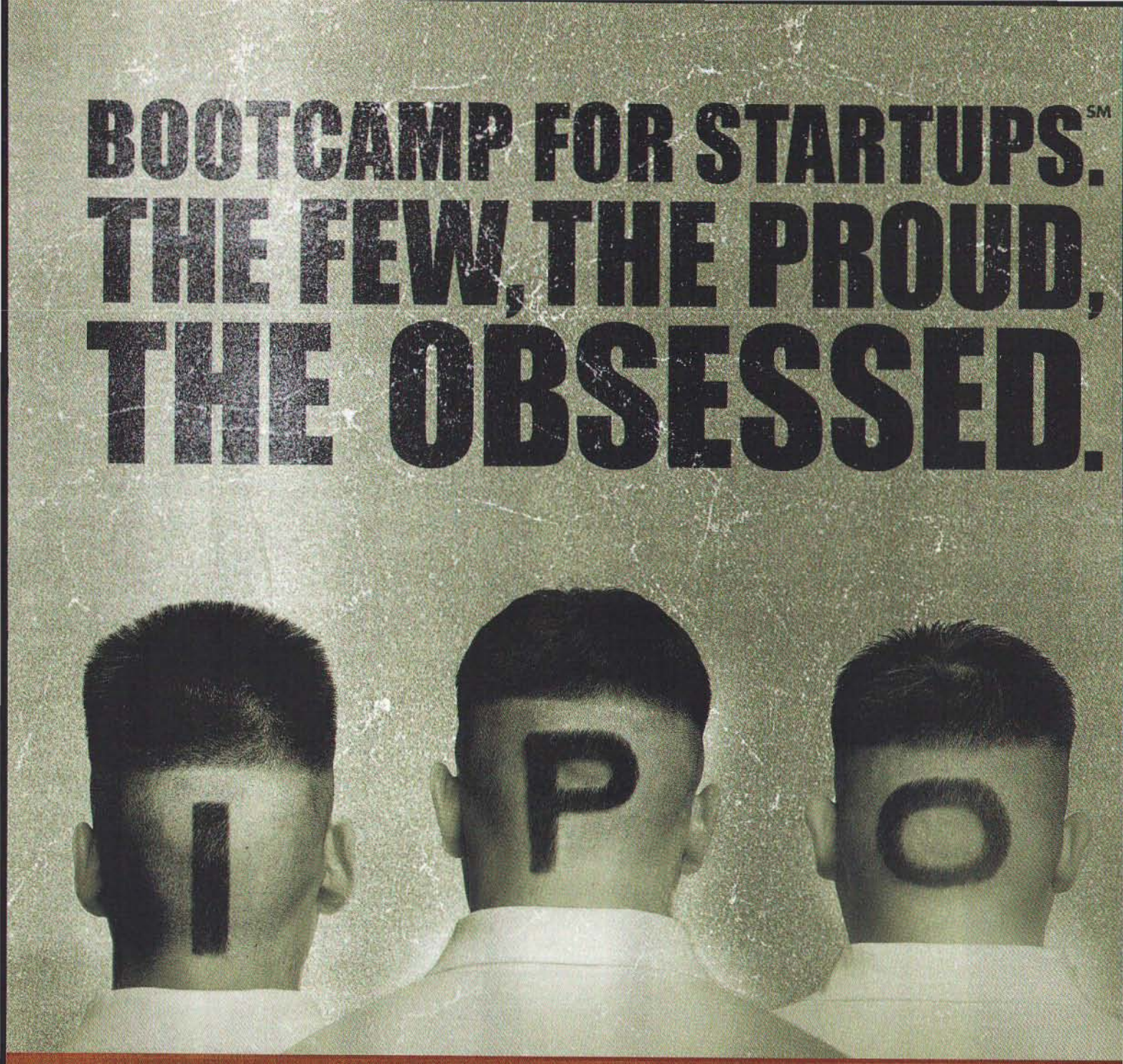
Figure 3. *Custom-generated PDF containing rotated text and stroked vector graphics.*

To understand what's going on, let's look at the above form input, one line at a time. The first five lines are encapsulated by <@> and </@> , which means we're writing raw PDF. The first line of raw PDF contains 0.85 0.85 0.85 RG, which sets the stroke color (in our default RGB color space) to 15% grey. In PDF, RGB channel values run from zero to 1.0, with 1.0 being brightest. Therefore, setting each channel to 0.85 makes for 15% grey (or 85% white).

The second line is more complicated. Basically, it says to set the default line end-cap style to rounded end-caps (that's what '1 J' means); set the default line width to 110 pixels (thus '110 w'); move the pen to page coordinates of 250,600 ('m' means "moveto"); draw a line to 250,600; and finally, stroke the path ('B'). In plain English:



BOOTCAMP FOR STARTUPS.℠
THE FEW, THE PROUD, THE OBSESSED.



SEATTLE, February 28–29 **NEW YORK, April 10–11**

garage.com™
we start up startups

Attention: Join Garage.com's two-day Bootcamp for Startups. Learn the fundamentals of taking your company from startup to IPO. Hear from the high tech industry's top investors, experts, and entrepreneurs. Gain invaluable information about raising capital, building a buzz, hiring top talent, and launching your product. At ease.

LOG ON TO **WWW.GARAGE.COM/BOOTCAMP** TO LEARN MORE & REGISTER TODAY.

Forbes CAPITALIST TOOL · Venture Law Group · Silicon Valley Bank · KPMG Consulting

IBM · Microsoft · ONVIA.com · StorageNetworks · EXODUS · mofo.com

"Draw a zero-length line, 110 pixels wide, at 250,600 with rounded end caps." Since the line has zero length, *only the end caps are drawn*. In this way, we spoof Acrobat into drawing a perfect circle, which looks filled (although really it's only stroked). This is an old Postscript hack for drawing filled circles. The reason we use it here is that PDF doesn't have a "draw circle" operator, just Bézier-curve operators. (I don't know about you, but I find the idea of drawing a circle using linked Bézier segments too painful to think about.)

Next, in the third line of raw PDF, we set the stroke color to 40% grey (60% white) with .6 .6 .6 RG.

In line four, we change the default stroke width to 18 pixels ('18 w'); move the pen to 72,72; draw a line to 72,720; another line to 540,720; a line to 540,72; and then we use the 's' command, which in PDF means "close this path and stroke it." This is how we produced a thick, dark-grey line going around the margins of the page.

Finally, in the fifth line we perform some matrix magic. Transform matrices, in PDF (as in Postscript) consist of six numbers, the first four of which specify scaling and rotation. To rotate the coordinate system, just make sure the first four numbers represent (consecutively) the cosine, sine, negated sine, and cosine of the desired angle. In our example, we used 90 degrees as the angle (hence the matrix values 0, 1, –1, and 0). The final two numbers in the transform matrix represent translation values in *x* and *y*. That is, we move the origin of space to whatever these numbers are (in this case, 144,520). Recall that the origin starts life at the lower left corner of the page.

Now when our text is drawn, it begins at 144,520 but runs at 90 degrees up the page. Since text is (by default) filled but not stroked, it gets drawn in black (the default fill color, which we have not changed).

Simple as pi.

Conclusion

We've covered an incredible amount of ground in a relatively small space. Not only have we shown how to serve dynamic PDF pages from a compact (80 lines or so) Perl script, but we've integrated a low-level "raw PDF" editor into the data-input form such that if the user wants to, he can produce sophisticated graphics effects inside the generated PDF page. (A "live" copy of this script, and others similar to it, can be found online at http://www.acroforms.com/dynamicPDF.html.)

Of course, many useful additions could still be made to our little script. A useful exercise would be to add such features as:

- Additional fonts, and a facility for the user to change font sizes on the fly.
- Automatic line-wrapping, with or without justification.
- Custom margin widths.
- Custom background colors. (This is just a matter of drawing a filled, colored rectangle large enough to cover the entire page.)
- Stylesheets.

You can probably think of lots of other features that would be nice to have. The good news is, now you're in a position to implement them, because (as you now know) dynamic PDF really isn't that hard. MT

NEW! Version 2.2

Scripter 2 with ScriptBASE

Tap the power of AppleScript with Main Event's Scripter!

For professionals and novices
Webmasters and solution providers

No matter what your experience, Scripter makes it easier!

BEGINNER AT SCRIPTING?

- Unique command-builders help you learn to assemble commands
- Built-in tools for experimenting
- Shortcuts to speed scripting

EXPERIENCED WITH APPLESCRIPT?

- Unmatched single-step interactive debugging
- Watch and modify global and local variables while stepping
- Debug messages sent to script applications
- Includes ScriptBase to integrate scripting scenarios

Scripter received MacWeek's 5-diamond rating – Twice! "Scripter's virtuoso display of AppleScripting savvy and debugging wizardry make it the best environment we've found for learning or creating AppleScript scripts." —MacWeek

Make AppleScript and your applications work for you!

Main Event
PO Box 21470
Washington, DC 20009
Tel: 202-298-9595
Sales: 800-616-8320
info@mainevent.com
www.mainevent.com

By Tim Monroe

Opening The Toolbox

Controlling QuickTime Movies With the Movie Toolbox

Introduction

In the previous two *QuickTime Toolkit* articles, we've focused almost exclusively on playing QuickTime movies using movie controllers. This is by far the easiest way to manage movies and to support the standard types of user interaction with movies. Indeed, some types of QuickTime movies cannot be presented to or managed by the user without a movie controller. But there are other types of QuickTime movies (most notably, standard linear movies) that can be played back without using a movie controller. In this article, we'll see how to accomplish this. In effect, our goal will be to provide all the capabilities that are provided by our basic application QTShell — for linear movies at least — but without the assistance of movie controllers. We'll show how to start and stop movies, provide basic editing services, and set various looping states.

Why might we want to do this? Well, there are situations where you want to play QuickTime movies but don't necessarily want the user to be able to manipulate those movies in the ways typically afforded by movie controllers. Perhaps the simplest example of this is found quite often in games, where you might play what's called a *cut movie* or a *cut scene* when the user completes one level and advances to the next. The cut movie serves as a transition from one level to the next, and you generally don't want the user interrupting or (heaven forbid!) editing your cut movie. So using the full-fledged movie controller interface is definitely not the best way to handle QuickTime movie playback here.

There are also more complicated examples of using controller-less QuickTime movies. For instance, you might want to use a linear QuickTime movie as a texture for a 3D object that's being displayed by your application. **Figure 1** shows a window of the application TextureEyes, which uses QuickDraw 3D to define and render three-dimensional objects. In this window, we've created a cube and set its surface texture to be a QuickTime movie.

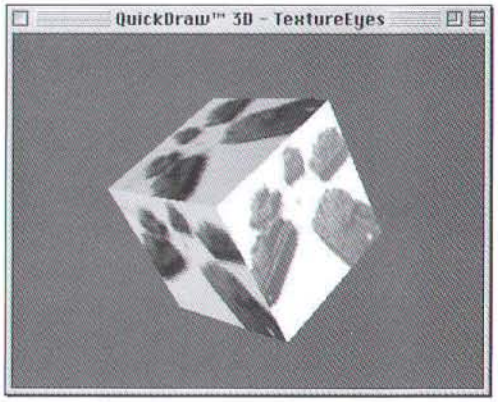


Figure 1. *A QuickTime movie used as a texture on a 3D object.*

Similarly, we might want to embed a QuickTime movie within another QuickTime movie. For instance, QuickTime VR allows us to draw into a panoramic node. We can use this capability to draw the individual frames of a QuickTime movie into a panorama, as shown in **Figure 2**. Here, the toy train on

Tim Monroe <monroe@apple.com> is a software engineer on Apple's QuickTime team. He is currently developing sample code and utilities for the QuickTime software development kit.

the table isn't part of the original panorama. Rather, we're drawing the frames of a standard QuickTime movie (one of those frames in shown in **Figure 3**) into the panorama, using the QuickTime VR application programming interfaces and some QuickDraw routines to drop out the black background. The net effect is that the train appears to travel in a circle on the table, thereby imparting some motion and sound to the otherwise static and silent panorama.

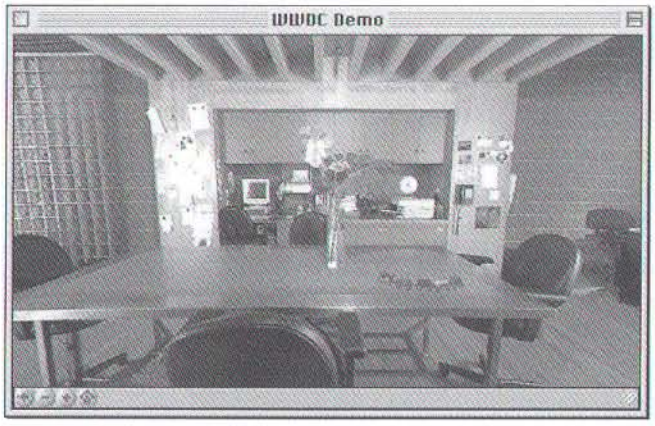


Figure 2. *A QuickTime movie embedded into a QuickTime VR panorama.*

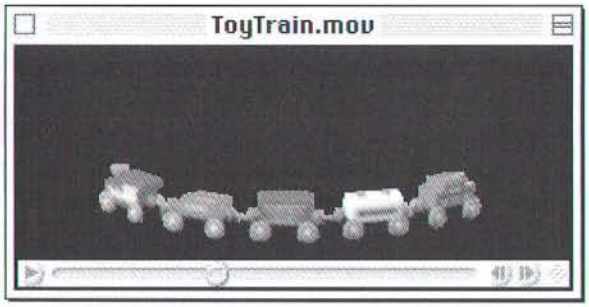


Figure 3. *The QuickTime movie embedded in Figure 2.*

We might also want to embed a linear QuickTime movie into another linear QuickTime movie, as shown in **Figure 4**.

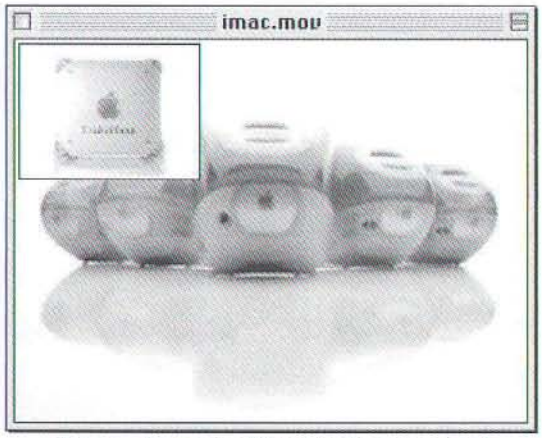


Figure 4. *A QuickTime movie embedded into a QuickTime movie.*

The point is not that we couldn't have used movie controllers to help us display and manage the cut scenes or the textured or embedded QuickTime movies in these sorts of cases. Rather, the point is that we don't need to do so. In the examples we've mentioned, we just need a way to manage the display of the movie frames at the appropriate time (and to play back the movie's sound track, if it has one). So in those cases the only services we really need are QuickTime's imaging, sound producing, and timing services. We don't need the robust user-interaction services provided by the standard linear movie controller.

It's really a question of levels: the movie controller interface for controlling movies is a higher-level interface than what we need to accomplish the tasks outlined above. Now we'll drop down a level and directly use the services of that part of QuickTime known as the *Movie Toolbox*. The Movie Toolbox is really the heart and soul of QuickTime. It provides services for opening movie files and extracting movies from them, playing movies, editing movies, creating movies, and much more. Indeed, the original *Inside Macintosh* documentation for the Movie Toolbox (that is, circa 1993) ran to well over 400 printed pages. To a large degree, programming with QuickTime is using the Movie Toolbox.

So here's what we want to accomplish in this article: we want to modify our basic application QTShell so that it can open and play back linear QuickTime movies without using movie controllers. Let's call this new application QTMooVToolbox, in honor of the fact that we'll be doing everything using the Movie

DATA RECOVERY: 800-440-1904

"Their incredibly kind staff focus on getting the job done as well as their customer's feelings. All experiences should be so pleasant."

–Neil Ticktin, Publisher
MacTech Magazine

7 Good Reasons to Choose DriveSavers

1. Fastest, most successful data recovery service available.
2. Recommended and certified by **all** drive companies to open the drive and maintain the warranty.
3. Advanced, proprietary recovery techniques.
4. 24-hour, onsite, and weekend services.
5. Retrieve recovered data instantly with DATAEXPRESS™ over high speed secured Internet lines.
6. Featured in Macworld, Mac Addict, MacWeek, Popular Mechanics; also by CNN, BBC, Forbes, and many others.
7. Federal and State Contracts.

DRIVESAVERS DATA RECOVERY

Since 1985

"We Can Save It!"

INTL: 415-382-2000
www.drivesavers.com

DriveSavers – Your Data Recovery Solution!

©2000 DRIVESAVERS, INC. 400 BEL MARIN KEYS BLVD., NOVATO, CA 94949 INTL: 415-382-2000 FAX: 415-883-0780

Toolbox. Since we're not using a movie controller, there will be no movie controller bar in our movie windows. A typical movie window will therefore look like the one in **Figure 5**.

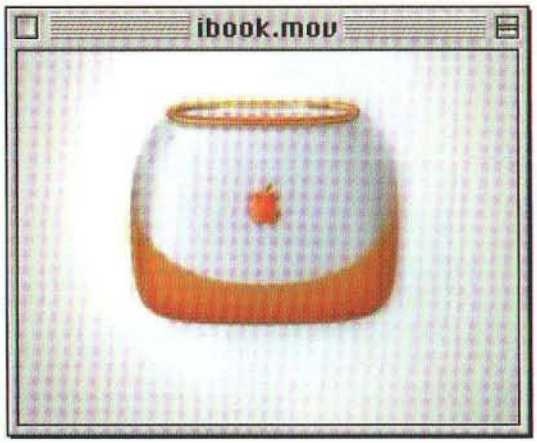


Figure 5. *A movie window displayed by QTMooVToolbox.*

As indicated above, we want to support the standard movie editing capabilities, so our application's Edit menu will be unchanged (though of course the code providing the editing capabilities will change quite a bit). We also want to provide the user a way to start and stop the movie and to set normal and palindrome looping. **Figure 6** shows the Test menu that we want to support.

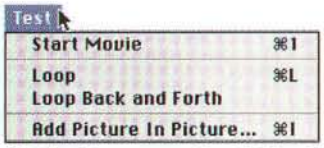


Figure 6. *The Test menu for QTMooVToolbox.*

The last menu item here ("Add Picture In Picture...") is particularly interesting. It allows us to embed one linear QuickTime movie inside of another linear QuickTime movie (as shown in **Figure 4**). As we'll see, it's fairly easy to use the Movie Toolbox to accomplish this.

Getting Started

Setting Up to Use QuickTime

Before moving forward with our QTMooVToolbox application, we need to back up a step or two to fill in a few details that we skipped over in the first two articles. In order to use any of the QuickTime services, we need to make sure that the QuickTime client software is available on the target machine. On Windows computers, we can do this by trying to initialize the QuickTime Media Layer, like this:

```
myErr = InitializeQTML(0L);
```

If InitializeQTML completes successfully (that is, if myErr is noErr), then we know that QuickTime is available. On the other hand, if InitializeQTML doesn't complete successfully, then either QuickTime or some other essential component is missing, in which case we will just exit the application (perhaps after informing the user that QuickTime needs to be installed).

On Macintosh computers, we can call the Gestalt function to determine whether QuickTime is available, as shown in Listing 1:

Listing 1: Determining whether QuickTime is available.

QTUtils_IsQuickTimeInstalled

```
Boolean QTUtils_IsQuickTimeInstalled (void)
{
  long    myAttrs;
  OSErr   myErr = noErr;
  myErr = Gestalt(gestaltQuickTime, &myAttrs);
  return(myErr == noErr);
}
```

On either Windows or Macintosh computers, once we've determined that QuickTime is indeed available in the current operating environment, we need to call the EnterMovies function, like so:

```
myErr = EnterMovies();
```

EnterMovies initializes the Movie Toolbox, performing any necessary set-up for our application (such as allocating any private storage it might need to use on our behalf). As with InitializeQTML, we will check the value returned by EnterMovies and exit the application if an error occurs.

When we are finished using QuickTime services — usually, when our application is about to terminate — we should undo the work done by EnterMovies and (on Windows) InitializeQTML. We can call the ExitMovies function to terminate our connection with the Movie Toolbox. On Windows, we also need to call TerminateQTML to unregister our application from the QuickTime Media Layer.

Using Application-Specific Data

Suppose now that our application is up and running, that we've successfully initialized QuickTime and the Movie Toolbox, and that the user has decided to open a movie file (perhaps using the Open command in the File menu). Our basic cross-platform framework calls the QTFrame_OpenMovieInWindow function to prompt the user for the movie file to open, create a window object record to hold basic information about the movie and the file it is contained in, open a new window on the screen, adjust the size of that window to fit the movie's natural size, and do some other useful set-up before the user can begin to interact with the movie.

QTFrame_OpenMovieInWindow also calls the function QTApp_SetupWindowObject, defined in the file ComApplication.c, to allow our application to perform any application-specific actions on the new movie, before it is displayed to the user. Both of our previous applications (namely, QTShell and QTController) have not put any code at all into QTApp_SetupWindowObject because the default configuration was sufficient for our purposes then. Here, for QTMooVToolbox,

we'll need to change that, for two reasons. First, we need to maintain some information for each open movie window beyond what is contained in the window object record. Second, since we are not using movie controllers in the QTMooVToolbox application, we need to do some additional work to prepare our movies for playback. Let's tackle the first task here and defer the second until a bit later, in "Preparing Movies For Playback".

We already have the apparatus in place to support adding application-specific data to our movie windows. As you may recall, the window object record includes the field fAppData, which is declared to be of type Handle. We put this field there precisely to provide ourselves a place to store any information that is specific to a particular application (since we didn't want to be changing the framework files all the time). For QTMooVToolbox, we'll define a structure of type ApplicationDataRecord that contains three fields:

```
typedef struct ApplicationDataRecord {
  MovieEditState  fMovieEditState;// the edit state of the movie
  Movie           fPIPMovie;          // the picture-in-picture movie
  Rect            fPIPRect;       // the picture-in-picture rectangle
} ApplicationDataRecord, *ApplicationDataPtr,
**ApplicationDataHdl;
```

We don't need to explain these fields quite yet (though the comments should give you a good enough idea of what they are for). But we do need to show how the application data record gets created and attached to the window object. We do this in the function QTApp_SetupWindowObject defined in Listing 2.

Listing 2: Setting up a window object.

QTApp_SetupWindowObject

```
void QTApp_SetupWindowObject (WindowObject theWindowObject)
{
  Movie                   myMovie = NULL;
  ApplicationDataHdl      myAppData = NULL;
  OSErr                   myErr = noErr;
  if (theWindowObject != NULL) {
    // get rid of the movie controller that's already been attached to this window object;
    // we do this to illustrate how to manage movies without using a movie controller
    if ((**theWindowObject).fController != NULL) {
      MCSetActionFilterWithRefCon((**theWindowObject)
                                  .fController, NULL, 0L);
    DisposeMovieController((**theWindowObject).fController);
      (**theWindowObject).fController = NULL;
    }
    // allocate an application-specific data record
            myAppData = (ApplicationDataHdl)NewHandleClear
                        (sizeof(ApplicationDataRecord));
    (**theWindowObject).fAppData = (Handle)myAppData;
    if (myAppData != NULL) {
      (**myAppData).fMovieEditState = NULL;
      (**myAppData).fPIPMovie = NULL;
      MacSetRect(&(**myAppData).fPIPRect, 0, 0, 0, 0);
    }
    // some lines omitted here
  }
}
```

Notice that the first thing we do in QTApp_SetupWindowObject is get rid of the movie controller that has already been created by the time QTApp_SetupWindowObject is called. That's because (as you already know) we want to illustrate how to manage movies without using a movie controller. We might as well just dispose of the controller, so we're sure it's not doing any work for us.

The second thing we do in QTApp_SetupWindowObject is call NewHandleClear to allocate an application data record and initialize its fields. (Of course, the initialization here is overkill, since NewHandleClear zeros the bytes in the newly-allocated block; nonetheless, it's useful to make this initialization explicit.) Then we set the fAppData field of the window object passed to QTApp_SetupWindowObject to the newly-allocated handle. This allows us, as we intended, to keep a copy of this record for each movie window.

The file ComFramework.c defines a couple of functions that facilitate retrieving the application-specific data attached to a movie. For instance, we can use the function QTFrame_GetAppDataFromWindowObject, defined in Listing 3, to retrieve the application-specific data associated with a window object.

Listing 3: Getting the application-specific data attached to a window object.

QTFrame_GetAppDataFromWindowObject

```
Handle QTFrame_GetAppDataFromWindowObject (WindowObject
theWindowObject)
{
  // make sure this is a window object belonging to our application
  if (!QTFrame_IsWindowObjectOurs(theWindowObject))
    return(NULL);
  return((**theWindowObject).fAppData);
}
```

Similarly, we can use the function QTFrame_GetAppDataFromWindow, defined in Listing 4, to retrieve the application-specific data attached to a particular movie window.

Listing 4: Getting the application-specific data attached to a movie window.

QTFrame_GetAppDataFromWindow

```
Handle QTFrame_GetAppDataFromWindow (WindowReference theWindow)
{
  WindowObject    myWindowObject = NULL;
  myWindowObject = QTFrame_GetWindowObjectFromWindow(theWindow);
  if (myWindowObject == NULL)
    return(NULL);
  return(QTFrame_GetAppDataFromWindowObject(myWindowObject));
}
```

We'll use these functions extensively throughout the rest of this article.

Updating the Application Framework

The basic framework for playing QuickTime movies that we've used for the QTShell and QTController sample applications makes extensive use of movie controller components to manage the movie playback and any user interaction with the movie. It uses movie controllers to help set the window size, handle movie editing, process movie-related events, and a handful of other important tasks. So, I must admit that I was more than a little worried when I set about to develop the QTMooVToolbox application for this article. After all, the very first application-specific thing we do (in Listing 2) is throw away the movie controller that the framework so kindly created and configured for us. How much of the entire framework was dependent upon work done by that movie controller? Or, asked another way, how much of the basic QTShell framework would we need to rewrite in order to play back movies using only the Movie Toolbox?

Surprisingly, very little had to change. In fact, only three functions needed reworking to handle the new assumption that the movie window might no longer have a movie controller associated with it. First of all, the QTFrame_SizeWindowToMovie function originally contained these lines:

```
myMC = (**theWindowObject).fController;
myMovie = (**theWindowObject).fMovie;

if (MCGetVisible(myMC))
  MCGetControllerBoundsRect(myMC, &myMovieBounds);
else
  GetMovieBox(myMovie, &myMovieBounds);
```

This just says: if the movie has a visible movie controller bar, then get the desired window size by calling MCGetControllerBoundsRect (which returns the rectangle that encloses the movie and the movie controller bar); otherwise, call GetMovieBox (which returns the rectangle that encloses just the movie). To allow for the case where there is no movie controller at all, we can rewrite the key lines like this:

```
GetMovieBox(myMovie, &myMovieBounds);
if (myMC != NULL)
  if (MCGetVisible(myMC))
    MCGetControllerBoundsRect(myMC, &myMovieBounds);
```

Second, the framework function QTFrame_HandleEditMenuItem also assumed that every movie window had a movie controller associated with it, in which case it could use movie controller functions like MCCut and MCCopy to handle the Edit menu items. In QTMooVToolbox, this is no longer true, so we need to provide a way for an application to intercept the Edit menu commands and do its own type of movie editing. We can do this very simply by inserting a call to QTApp_HandleMenu before QTFrame_HandleEditMenuItem handles the menu items. If QTApp_HandleMenu returns the value true to indicate that it has handled the menu item, then we just skip the rest of QTFrame_HandleEditMenuItem. We'll see later, in "Editing Movies", how our application actually handles the Edit menu items; for the moment, at least we have successfully provided a way to prevent the framework from trying to handle them using movie controller functions.

The last important change we need to make to our basic framework is to provide a way for both our Macintosh and Windows applications to tell the Movie Toolbox to play any movies that are open. When we're using QuickTime, we don't just start a movie playing and then forget about it, expecting perhaps that it will play through until the end and then stop. Instead, from time to time we need to explicitly allocate some processor time to the Movie Toolbox so that it can do whatever it needs to do in order to keep the movie playing. This periodic allocation of processor time to the Movie Toolbox is known as *tasking* a movie, and we can accomplish it by calling the MoviesTask function over and over until the movie is done playing. So what we need to do now is provide a way for QTMooVToolbox to call MoviesTask periodically. (When we are using movie controllers, we don't need to ever call MoviesTask, because the MCIsPlayerEvent function calls it internally for us.)

A standard way to do this, on the Macintosh, is to insert a call to MoviesTask in the application's main event loop. For several reasons (which we don't need to go into), we'll adopt a slightly different technique in QTMooVToolbox; we'll call MoviesTask in response to null events (which occur often enough to keep the movie playing smoothly). For Windows applications, however, this is slightly trickier, since there is no Windows equivalent of the Macintosh null or idle event (and there is no main event loop). Probably the best solution for Windows applications would be to install a timer that calls MoviesTask periodically. For the moment, however, we'll adopt a different (and much simpler) approach. Our basic Windows framework already calls the function QTApp_HandleEvent every time a movie window receives a message to process. So we'll just look for null events there and call QTApp_Idle (which in turn calls MoviesTask) for every open movie window. Listing 5 shows the changes we can make to QTApp_HandleEvent to keep our Windows movies running smoothly.

Listing 5: Getting Windows applications to call QTApp_Idle.

QTApp_HandleEvent

```
Boolean QTApp_HandleEvent (EventRecord *theEvent)
{
  Boolean           myIsHandled = false;
#if TARGET_OS_WIN32
  WindowReference   myWindow = NULL;
  if (theEvent != NULL)
    if (theEvent->what == nullEvent) {
      myWindow = QTFrame_GetFrontMovieWindow();
      while (myWindow != NULL) {
        QTApp_Idle(myWindow);
        myWindow = QTFrame_GetNextMovieWindow(myWindow);
      }
    }
#endif
  return(myIsHandled);
}
```

Keep in mind that these strategies might not work for more complicated or more specialized applications. On the Macintosh, for instance, an application might be doing so much processing that there are not enough null events generated to keep the movie playing smoothly. Conversely, our Windows strategy might consume too much processor time with its almost constant calls to QTApp_Idle. So QTMooVToolbox most certainly does not provide a universal solution to the problem of when and how to call MoviesTask. The strategies that we've adopted here are designed only to keep the movies running smoothly, with a minimum of changes to our basic QuickTime framework.

Preparing Movies For Playback

So far, our application can open QuickTime movie files, read the movies from those files, and create appropriately-sized windows to hold those movies. Are we finally ready to begin playing the movies? In other words, when the user indicates that he or she wants the movie to begin playing, can we just start the ball rolling by calling the StartMovie function and then let our framework task the movie periodically?

The answer here is: yes and no. We could indeed just call StartMovie, and most movies would start playing pretty much immediately, as desired. Other movies might require a small but noticeable amount of time to perform some necessary set-up activities, such as allocating buffers to hold data from the movie file and setting up various QuickTime components to handle the process of decompressing the data in the file. This set-up is unavoidable; however, if our application doesn't start the movie playing immediately, we can perform this set-up right after the user opens the movie and before the user actually starts playing the movie, so that it is effectively unnoticeable. This process is called *prerolling* the movie.

For some movies — in particular, for streamed movies located remotely on the web or on file servers — prerolling alone is not sufficient to guarantee an optimal user experience. That's because some actions need to occur even before the movie can successfully be prerolled. QuickTime needs to set up connections to the remote servers and negotiate one or more protocols for exchanging data; it also needs to get information about the types of data in those remote movies. This process is called *preprerolling* the movie (because it needs to occur prior to the prerolling process). Unless we prepreroll movies stored remotely, the movie playback would be annoyingly choppy and some of the data in those movies (for instance, the sound) might not play correctly at all.

So we need a more general method of starting up movies that will accommodate both local and remote QuickTime movies. We need to prepreroll and preroll our movies. Let's consider these operations in reverse order.

Prerolling Movies

To preroll a movie, we can use the Movie Toolbox function PrerollMovie. As indicated above, PrerollMovie does whatever it can to prepare the movie for immediate playback once the movie is started. This involves determining what kinds of data are in the movie and setting up the appropriate software components to handle those kinds of data. Prerolling might also involve actually reading some of the movie's data and getting the relevant decompressors ready to decompress that data. Because of this, PrerollMovie needs to know the current location in the movie and the rate at which the movie is to be played. We can get the current location in the movie by calling GetMovieTime, and we can get the movie's default playback rate by calling GetMoviePreferredRate. So we can preroll a movie by executing these lines of code:

```
myTimeValue = GetMovieTime(myMovie, NULL);
myPlayRate = GetMoviePreferredRate(myMovie);
PrerollMovie(myMovie, myTimeValue, myPlayRate);
```


Valentina

object-relational database engine

The fastest database engine for the MacOS

It operates 100's and sometimes a 1000 times faster than other systems

Valentina - scriptable database application **$49**
- Includes special features for Web Developers.

Valentina SDK for C++ developers **$499**
- MacOS and Windows 95/98/NT versions; cross-platform databases.
- Support for almost any type of field: bool,byte,short,long,float,double,date,time,string BLOB,TEXT, Picture.
- Support for indexing and sorting of international languages.
- SQL random-access cursors.
- Multi-threading capable.
- Record locking.
- No runtime fees.

Valentina for REALbasic plugin **$199**
Valentina for Macromedia Director Xtra **$199/299**
Valentina for WebSiphon **$299**

Make your application's database operations blazingly fast!

Order Directly from Our Web Site — **www.paradigmasoft.com** Hosted by MacServe.net

Download full featured evalution version

If for some reason you want to change the current location in the movie or the movie's default playback rate, you should do it before calling PrerollMovie so that you don't invalidate the work done by PrerollMovie. You can set the current location in the movie by calling SetMovieTime, and you can set the default playback rate by calling SetMoviePreferredRate. The default playback rate is the rate that StartMovie uses when it starts a movie playing. When you open a movie from a movie file, the actual playback rate is 0 (that is, the movie is not playing).

Preprerolling Movies

To preproll a movie, we can use the Movie Toolbox function PrePrerollMovie, which establishes any required network connections between the computer running our application and the computer that contains the movie being preprerolled. If the movie resides on the same computer as the application (that is, if the movie is a local movie), then PrePrerollMovie has nothing to do and it returns immediately. As a result, there is no harm in calling PrePrerollMovie for every movie our application opens.

On the other hand, if the movie being preprerolled is not a local movie, then preprerolling it can take a significant amount of time, particularly if the local computer needs to access the remote computer over a dial-up line. To keep your application from stalling while QuickTime negotiates a connection with the remote computer, you can call PrePrerollMovie asynchronously, in which case PrePrerollMovie returns almost immediately and then executes an application-defined completion function when the preprerolling is finished. PrePrerollMovie has the same calling interface as PrerollMovie, except that you must also specify a *movie preprerolling completion routine* and an optional reference constant that is passed to the completion routine:

```
myPlayRate = GetMoviePreferredRate(myMovie);
PrePrerollMovie(myMovie, 0, myPlayRate,
gMoviePPRollCompleteProc, (void *)theWindowObject);
```

Here, gMoviePPRollCompleteProc is a universal procedure pointer (UPP) to the movie preprerolling completion routine. QTMooVToolbox allocates this UPP when it starts up, in the function QTMT_Init defined in Listing 6.

Listing 6: Allocating a routine descriptor for QTMT_MoviePrePrerollCompleteProc.

QTMT_Init

```
void QTMT_Init (void)
{
  // allocate a routine descriptor for our preprerolll completion routine
  gMoviePPRollCompleteProc =
NewMoviePrePrerollCompleteProc(QTMT_MoviePrePrerollCompleteProc);
}
```

When the preprerolling is complete, the Movie Toolbox executes our movie preprerolling completion function, defined in Listing 7.

Listing 7: Responding when a movie is done preprerolling.

QTMT_MoviePrePrerollCompleteProc

```
PASCAL_RTN void QTMT_MoviePrePrerollCompleteProc (Movie
theMovie, OSErr thePrerollErr, void *theRefCon)
{
#pragma unused(thePrerollErr)
  WindowObject        myWindowObject =
(WindowObject)theRefCon;
  ApplicationDataHdl myAppData = NULL;
  TimeValue          myTimeValue;
  Fixed              myPlayRate;

  myAppData =
(ApplicationDataHdl)QTFrame_GetAppDataFromWindowObject
                                        (myWindowObject);
  if ((theMovie == NULL) || (myWindowObject == NULL) ||
                                    (myAppData == NULL))
    return;
  // preroll the movie, so that it's ready to play when we call StartMovie
  myTimeValue = GetMovieTime(theMovie, NULL);
  myPlayRate = GetMoviePreferredRate(theMovie);
  PrerollMovie(theMovie, myTimeValue, myPlayRate);
  // for the picture-in-picture movie, start it playing in a loop
  if (theMovie == (**myAppData).fPIPMovie) {
    QTMT_SetMovieLoopState(theMovie, kNormalLooping);
    StartMovie(theMovie);
  }
}
```

As you can see, our movie preprerolling completion function is quite simple. It really just prerolls the movie, thereby making the movie ready for playback. For some movies (namely, for QTMooVToolbox's "picture-in-picture movies"), it also starts the movies playing in a loop. We'll discuss picture-in-picture movies later (in "Playing Picture-In-Picture Movies").

Note that preprerolling and prerolling are required only for movies that we are managing directly, using the Movie Toolbox. If we use the movie controller interface for managing movies, the movie controller automatically preprerolls and prerolls the movies for us.

Playing Movies

Now we are finally ready to start our movie playing. Assuming that a movie has already been preprerolled and prerolled, we can set it in motion by setting its rate to a non-zero value, using the SetMovieRate function. Or, we can simply call the StartMovie function, which sets the movie rate to the default (or "preferred") movie rate. QTMooVToolbox defines the function QTMT_StartMovie (shown in Listing 8) to start a movie.

Listing 8: Starting a movie playing.

QTMT_MoviePrePrerollCompleteProc

```
void QTMT_StartMovie (Movie theMovie)
{
  if (theMovie == NULL)
    return;

  // if we aren't looping and we're at the end of the movie,
  // return to the beginning of the movie
  if (!QTMT_IsMovieLooping(theMovie))
    if (GetMovieTime(theMovie, NULL) ==
                                GetMovieDuration(theMovie))
      GoToBeginningOfMovie(theMovie);

  StartMovie(theMovie);
}
```

QTMT_StartMovie checks to see whether a non-looping movie has reached its end; if it has, QTMT_StartMovie calls the Movie Toolbox function GoToBeginningOfMovie to reset the current movie time to the beginning of the movie. If we had been playing the movie using a movie controller, the movie controller would have done this for us. Since we're programming at a slightly lower level, we need to do some of this housekeeping ourselves.

Similarly, when we are playing a non-looping movie using the Movie Toolbox and the movie reaches its end, its rate is not automatically set to 0. Again, this is some minor housekeeping that we need to take care of ourselves. In this case, we'll do the appropriate check in our QTApp_Idle routine, defined in Listing 9.

Listing 9: Performing idle-time tasks.

QTMT_MoviePrePrerollCompleteProc

```
void QTApp_Idle (WindowReference theWindow)
{
  GrafPtr                mySavedPort;
  WindowObject           myWindowObject = NULL;
  ApplicationDataHdl     myAppData = NULL;
  GetPort(&mySavedPort);
  MacSetPort(QTFrame_GetPortFromWindowReference(theWindow));
  myWindowObject =
              QTFrame_GetWindowObjectFromWindow(theWindow);
  myAppData =
            (ApplicationDataHdl)QTFrame_GetAppDataFromWindow
                                              (theWindow);
  if ((myWindowObject != NULL) && (myAppData != NULL)) {
    if (!QTMT_IsMovieLooping((**myWindowObject).fMovie))
      if (GetMovieTime((**myWindowObject).fMovie, NULL) ==
                GetMovieDuration((**myWindowObject).fMovie))
        if (GetMovieRate((**myWindowObject).fMovie) > 0)
          SetMovieRate((**myWindowObject).fMovie, 0);
    if ((**myWindowObject).fMovie != NULL)
      MoviesTask((**myWindowObject).fMovie, DoTheRightThing);

    if ((**myAppData).fPIPMovie != NULL)
      QTMT_DrawPictureInPictureMovie(myWindowObject);
  }
  MacSetPort(mySavedPort);
}
```

QTApp_Idle does three main things. It sets the movie rate of the movie in the specified window to 0 if it isn't looping and it has played to its end; it tasks the movie; and it draws any picture-in-picture movie in the specified window. (Once again, we'll discuss picture-in-picture movies at more length later.) As you can see, MoviesTask takes two parameters; these are the movie to be tasked and a maximum number of milliseconds that MoviesTask should spend doing its work. Movies.h defines the constant DoTheRightThing as 0, which means to service the specified movie once and then return.

So, at last, we have at hand all the essential elements of playing movies using the Movie Toolbox. First, we open a movie file and load the movie from it (using the same code that we used when playing movies using movie controllers). Then, we prepreroll and preroll that movie. Then, we start the movie playing by calling StartMovie. And finally, we periodically call MoviesTask to give the movie some time to play. It's okay to call MoviesTask on a movie that has finished playing, so in QTApp_Idle we don't bother to check whether the movie is actually playing or not. In fact, we shouldn't do that, since we need to call MoviesTask on a movie while it is preprerolling as well as while it is playing.

Editing Movies

Editing movies is one of the tasks at which movie controllers excel. A movie controller keeps track of a movie's current selection and allows us (for instance) to cut that selection from that movie by calling the function MCCut, like this:

```
case IDM_EDITCUT:
  myEditMovie = MCCut(myMC);
  break;
```

As you can see, MCCut also returns to us a movie that is the cut selection, which we can put on the scrap (to make it available for subsequent pasting) by calling PutMovieOnScrap. Moreover, the movie controller keeps track of the current *edit state* of the movie that exists before each editing operation, so that it can restore that state if we subsequently call MCUndo. A movie's current edit state consists of all the information that describes the movie's contents, such as the sound and video data in the movie; the current edit state also includes the movie's current selection.

The Movie Toolbox provides a set of high-level movie editing functions, which are just as easy to use as the functions provided by movie controllers. For instance, to cut the current movie selection, we can call the CutMovieSelection function.

```
case IDM_EDITCUT:
  myEditMovie = CutMovieSelection(myMovie);
  break;
```

Similarly, we can use the ClearMovieSelection function to clear the current movie selection (that is, remove it from the movie without retaining the removed data).

```
case IDM_EDITCLEAR:
  ClearMovieSelection(myMovie);
  break;
```

Moreover, the Movie Toolbox provides the SetMovieSelection function, which we can use to handle the Select All menu item in the Edit menu, like this:

```
case IDM_EDITSELECTALL:
  SetMovieSelection(myMovie, 0L, GetMovieDuration(myMovie));
  break;
```

There is one complication here: if we want to provide the standard movie editing behavior without using a movie controller, then we also need to keep track of the movie's current edit state whenever we perform any editing operations. As we've seen, the application data record for QTMooVToolbox contains a field fMovieEditState, of type MovieEditState, which we'll use to hold the current edit state. Before we call any Movie Toolbox function that changes the contents of a movie, we can call the NewMovieEditState function to create a new movie edit state, as in this snippet from QTApp_HandleMenu:

```
if ((theMenuItem == IDM_EDITCUT) ||
    (theMenuItem == IDM_EDITPASTE) ||
    (theMenuItem == IDM_EDITCLEAR)) {
  if ((**myAppData).fMovieEditState != NULL)
    DisposeMovieEditState((**myAppData).fMovieEditState);
  (**myAppData).fMovieEditState = NewMovieEditState(myMovie);
}
```

Notice that we need to dispose of any existing edit state (by calling DisposeMovieEditState) before creating a new one. Otherwise, we'd be leaking memory.

If the user selects the Undo menu item in the Edit menu, we need to reinstate the movie edit state that we have stored in our application data record. We do this by calling UseMovieEditState, like this:

```
case IDM_EDITUNDO:
  if ((**myAppData).fMovieEditState == NULL)
    break;
  UseMovieEditState(myMovie, (**myAppData).fMovieEditState);
  DisposeMovieEditState((**myAppData).fMovieEditState);
  (**myAppData).fMovieEditState = NULL;
  break;
```

Once we have used an edit state, we want to dispose of it and set the stored edit state reference to NULL so that we don't reinstate the same edit state twice. QTMooVToolbox supports only one level of undo, just like the standard linear movie controller. In theory, however, you can NewMovieEditState and UseMovieEditState to support multiple levels of undo. Implementing this feature is left as an exercise for the reader.

There is one final complication to consider. If the user selects the entire movie and then cuts the current selection, the resulting movie is empty. It still exists, but it has no video or audio data in it. Since the movie has no video data, the movie box is empty. Now, if we had performed this cut using movie controller functions, the movie controller would have detected that the size of the movie had changed and would have notified our application by sending it the mcActionControllerSizeChanged movie controller action; in response to this action we would have called our application's QTFrame_SizeWindowToMovie function. But we didn't use movie controller functions to perform the cut, so we need to handle this possibility ourselves.

The simplest way to handle this situation is just to call QTFrame_SizeWindowToMovie whenever we perform any editing operation that might have changed the size of the movie, like this:

```
if ((theMenuItem == IDM_EDITUNDO) || (theMenuItem ==
IDM_EDITCUT) || (theMenuItem == IDM_EDITPASTE) ||
(theMenuItem == IDM_EDITCLEAR)) {
  QTFrame_SizeWindowToMovie(myWindowObject);
  (**myWindowObject).fIsDirty = true;
}
```

We now have all the pieces that we need to handle the Edit menu commands without using movie controllers. Listing 10 shows the QTApp_HandleMenu function for QTMooVToolbox. (For readability, I've cut out the Test menu handling.)

Listing 10: Handling Edit menu items using the Movie Toolbox.

QTApp_HandleMenu

```
Boolean QTApp_HandleMenu (UInt16 theMenuItem)
{
  WindowObject          myWindowObject = NULL;
  ApplicationDataHdl    myAppData = NULL;
  Movie                 myMovie = NULL;
  Movie                 myEditMovie = NULL;
  Boolean               myIsHandled = false;

  myWindowObject = QTFrame_GetWindowObjectFromFrontWindow();
  if (myWindowObject != NULL) {
    myAppData =
      (ApplicationDataHdl)QTFrame_GetAppDataFromWindowObject
                                          (myWindowObject);
    myMovie = (**myWindowObject).fMovie;
  }

  if ((myWindowObject == NULL) || (myAppData == NULL) ||
                                          (myMovie == NULL))
    return(myIsHandled);

  // before the Cut, Paste, and Clear commands, get the current edit state,
  // after disposing of the currently-saved edit state
  if ((theMenuItem == IDM_EDITCUT) || (theMenuItem ==
      IDM_EDITPASTE) || (theMenuItem == IDM_EDITCLEAR)) {
    if ((**myAppData).fMovieEditState != NULL)
      DisposeMovieEditState((**myAppData).fMovieEditState);
    (**myAppData).fMovieEditState = NewMovieEditState(myMovie);
  }

  switch (theMenuItem) {

    // Edit menu items
    // Here we are overriding the framework's Edit menu handling
    // (which uses movie controllers).

    case IDM_EDITUNDO:
      // restore the movie using the currently-saved edit state, then dispose of it
      if ((**myAppData).fMovieEditState == NULL)
        break;

      UseMovieEditState(myMovie, (**myAppData).fMovieEditState);
      DisposeMovieEditState((**myAppData).fMovieEditState);
      (**myAppData).fMovieEditState = NULL;
      break;

    case IDM_EDITCUT:
      myEditMovie = CutMovieSelection(myMovie);
      break;

    case IDM_EDITCOPY:
      myEditMovie = CopyMovieSelection(myMovie);
      break;

    case IDM_EDITPASTE:
      myEditMovie = NewMovieFromScrap(newMovieActive);
      if (myEditMovie != NULL) {
        PasteMovieSelection(myMovie, myEditMovie);
        DisposeMovie(myEditMovie);
        myEditMovie = NULL;
      }
      break;

    case IDM_EDITCLEAR:
      ClearMovieSelection(myMovie);
      break;

    case IDM_EDITSELECTALL:
      SetMovieSelection(myMovie, 0L, GetMovieDuration(myMovie));
      break;

    // Test menu items omitted here

    default:
      break;

  } // switch (theMenuItem)

  // place any cut or copied movie segment onto the scrap
  if (myEditMovie != NULL) {
    PutMovieOnScrap(myEditMovie, 0L);
    DisposeMovie(myEditMovie);
  }

  // after any commands that might have changed the movie, sync up the movie
  // window size and mark the window as dirty
  if ((theMenuItem == IDM_EDITUNDO) || (theMenuItem ==
          IDM_EDITCUT) || (theMenuItem == IDM_EDITPASTE) ||
                          (theMenuItem == IDM_EDITCLEAR)) {
    QTFrame_SizeWindowToMovie(myWindowObject);
    (**myWindowObject).fIsDirty = true;
  }

  // for any Edit menu command, indicate that we handled it
```

```
  if (MENU_ID(theMenuItem) == kEditMenuResID)
    myIsHandled = true;

  return(myIsHandled);
}
```

All in all, handling the Edit menu items using Movie Toolbox functions is not too much more complicated than doing it using the movie controller functions. We need to manage the movie edit states ourselves, and we need to make sure to resize a changed movie. But otherwise the code is quite reminiscent of the movie controller editing code.

Looping Movies

In the previous *QuickTime Toolkit* article, you might recall, we saw how to read a movie's looping state from some user data stored in the movie file. If a movie file contains a user data item of type 'LOOP', then the data in that item determines whether the looping state is normal looping (data is 0) or palindrome looping (data is 1). If there is no such user data item in the movie file, then we assume that the movie does not loop. To read a movie's looping state, we defined the function QTUtils_GetMovieFileLoopingInfo, which returns one of three constants defined in our header file QTUtilities.h:

```
enum {
  kNormalLooping              = 0,
  kPalindromeLooping          = 1,
  kNoLooping                  = 2
};
```

To set a movie's looping state based on the looping state stored in the file, we called the MCDoAction function, passing the two movie controller actions mcActionSetLooping and mcActionSetLoopIsPalindrome with appropriate combinations of true and false. Now we want to see how to accomplish the same thing without using these movie controller actions.

To manipulate a movie's looping state without using movie controller functions, we need to work with the movie's *time base*, which defines the time coordinate system of the movie. As *Inside Macintosh* nicely phrases it, we can think of a movie's time base like a vector that defines the direction and velocity of time for a movie. A movie's time base also defines the current movie time. So it makes sense that time bases are associated with looping: normal looping involves simply resetting the current movie time to 0 once the movie time reaches the end, while palindrome looping involves reversing the direction of time while keeping the velocity the same.

The Movie Toolbox creates a time base for us when we load a movie. Thereafter, we can retrieve a movie's time base by calling the GetMovieTimeBase function. The movie's looping state is determined by some of the *time base control flags*, a 32-bit value which we can get and set by calling the functions GetTimeBaseFlags and SetTimeBaseFlags. Movies.h defines these two time base flags related to looping:

```
enum {
  loopTimeBase                = 1,
  palindromeLoopTimeBase      = 2
};
```

These constants are interpreted as bit masks for the time base control flags. In other words, you should think of loopTimeBase as 1L<<0 and palindromeLoopTimeBase as 1L<<1. As you might have guessed, we can set a movie's looping state by getting the current time base control flags, setting the appropriate bits on or off, and then setting the revised time base control flags. Here's how we would enable palindrome looping:

```
TimeBase    myTimeBase = GetMovieTimeBase(theMovie);
long        myFlags = GetTimeBaseFlags(myTimeBase);

myFlags |= loopTimeBase;
myFlags |= palindromeLoopTimeBase;
SetTimeBaseFlags(myTimeBase, myFlags);
```

For optimal playback performance, however, we should take one further step, which is to set the movie's *playback hints* to reflect the desired looping state. A movie's playback hints specify one or more optimizations that the Movie Toolbox should use during movie playback. Once again, the playback hints are defined using individual bits in a 32-bit long word. These two bits are defined for setting movie looping hints:

```
enum {
  hintsLoop             = 1 << 1,
  hintsPalindrome       = 1 << 9
};
```

The Movie Toolbox provides the SetMoviePlayHints function for setting a movie's playback hints, but there is no corresponding GetMoviePlayHints function to get the current playback hints. Instead, SetMoviePlayHints requires us to pass it a 32-bit mask that determines which of the bits in the 32-bit play hints long word are to be considered. For example, to set the palindrome looping play hint on, we could do this:

```
SetMoviePlayHints(theMovie, hintsLoop, hintsLoop);
SetMoviePlayHints(theMovie, hintsPalindrome, hintsPalindrome);
```

These two lines of code set both the looping and palindrome looping play hints to be active. (Remember that looping must be enabled in order for palindrome looping to be enabled.) And to set the movie play hints to disable any looping, we could execute this line:

```
SetMoviePlayHints(theMovie, 0L, hintsLoop | hintsPalindrome);
```

So, putting this all together, we see that to set the looping state of a movie, we need to set the time base control flags appropriately; we also should set the movie's playback hints to optimize playback performance. Listing 11 defines our complete QTMT_SetMovieLoopState for setting a movie's looping state.

Listing 11: Setting a movie's looping state.

QTMT_SetMovieLoopState

```
void QTMT_SetMovieLoopState (Movie theMovie, UInt8
theLoopState)
{
  TimeBase      myTimeBase;
  long          myFlags = 0L;

  if (theMovie == NULL)
    return;
```

```
  myTimeBase = GetMovieTimeBase(theMovie);
  myFlags = GetTimeBaseFlags(myTimeBase);

  switch (theLoopState) {

    case kNormalLooping:
      SetMoviePlayHints(theMovie, hintsLoop, hintsLoop);
      SetMoviePlayHints(theMovie, 0L, hintsPalindrome);
      myFlags |= loopTimeBase;
      myFlags &= ~palindromeLoopTimeBase;
      break;

    case kPalindromeLooping:
      SetMoviePlayHints(theMovie, hintsLoop, hintsLoop);
      SetMoviePlayHints(theMovie, hintsPalindrome,
                                          hintsPalindrome);
      myFlags |= loopTimeBase;
      myFlags |= palindromeLoopTimeBase;
      break;

    case kNoLooping:
    default:
      myFlags &= ~loopTimeBase;
      myFlags &= ~palindromeLoopTimeBase;
      SetMoviePlayHints(theMovie, 0L, hintsLoop |
                                          hintsPalindrome);
      break;
  }

  SetTimeBaseFlags(myTimeBase, myFlags);
}
```

For certain purposes, we sometimes want to know whether a movie is looping and whether it is palindrome looping. Based on what we've learned so far, we can easily define the function QTMT_IsMovieLooping shown in Listing 12, which returns a Boolean value that indicates whether looping (either normal or palindrome) is enabled for the specified movie.

Listing 12: Determining whether a movie is looping.

QTMT_IsMovieLooping

```
Boolean QTMT_IsMovieLooping (Movie theMovie)
{
  TimeBase    myTimeBase;
  long        myFlags = 0L;

  if (theMovie == NULL)
    return(false);

  myTimeBase = GetMovieTimeBase(theMovie);
  myFlags = GetTimeBaseFlags(myTimeBase);

  return((Boolean)(myFlags & loopTimeBase));
}
```

And we can easily define the function QTMT_IsMoviePalindromeLooping, shown in Listing 13, to tell us whether palindrome looping is enabled for the specified movie.

Listing 13: Determining whether a movie is palindrome looping.

QTMT_IsMoviePalindromeLooping

```
Boolean QTMT_IsMoviePalindromeLooping (Movie theMovie)
{
  TimeBase    myTimeBase;
  long        myFlags = 0L;

  if (theMovie == NULL)
    return(false);

  myTimeBase = GetMovieTimeBase(theMovie);
  myFlags = GetTimeBaseFlags(myTimeBase);

  return((Boolean)((myFlags & loopTimeBase) &&
                        (myFlags & palindromeLoopTimeBase)));
}
```

Our application QTMooVToolbox uses these functions to set the appropriate check marks on the "Loop" and "Loop Back And Forth" menu items in the Test menu, like this:

```
QTFrame_SetMenuItemCheck(myMenu, IDM_SET_LOOPING,
QTMT_IsMovieLooping(myMovie) &&
!QTMT_IsMoviePalindromeLooping(myMovie));
QTFrame_SetMenuItemCheck(myMenu, IDM_SET_PALINDROME_LOOPING,
QTMT_IsMoviePalindromeLooping(myMovie));
```

Playing Picture-In-Picture Movies

Last Christmas, I was hoping that Santa Claus would bring me a new television set, perhaps a bit larger than the one I have now, and possibly also with the so-called "picture-in-picture" capability, where you can tune into a second channel that's displayed in a smaller box in the corner of the main picture. Well, Santa wasn't that generous this year, so I decided to see whether QuickTime could help satisfy my craving to watch two video sources at once. (See **Figure 4** once again.) It turns out that it's surprisingly straightforward to use the Movie Toolbox to embed one movie inside of another movie. Let's call the movie that we are embedding the "picture-in-picture movie" and the movie we're embedding it into the "main movie".

In the broadest outlines, we want to open the movie file that contains the picture-in-picture movie, set that movie's graphics port to be the window that already contains the main movie, and then set the picture-in-picture movie's enclosing rectangle to fit within the main movie window. Then we need to ensure that the main movie doesn't draw on top of the picture-in-picture movie; we do this by setting the main movie's display clipping region to exclude the rectangle occupied by the picture-in-picture movie. Finally, we need to task both movies in order to update the movie images and play the movies' sound. Let's see how to do these things.

Setting the Picture-In-Picture Movie Geometry

Since we want to draw the picture-in-picture movie in the same window as the main movie — and indeed on top of the main movie — we need to set the graphics port of the picture-in-picture movie to the same graphics port used by the main movie. We can do that quite easily, like this:

```
GetMovieGWorld(myMainMovie, &myGWorld, NULL);
SetMovieGWorld(myPIPMovie, myGWorld, NULL);
```

But, we don't want to draw the picture-in-picture movie at its natural size. Instead, we want it to occupy some smaller portion of the main movie. For simplicity, we'll set the picture-in-picture movie to have a width that is one-third the width of the main movie, and a height that is one-third the height of the main movie.

```
GetMovieBox(myMainMovie, &myRect);

myValue = (myRect.bottom - myRect.top) / 3;
myRect.top += myValue;
```

```
myRect.bottom -= myValue;

myValue = (myRect.right - myRect.left) / 3;
myRect.left += myValue;
myRect.right -= myValue;

SetMovieBox(myPIPMovie, &myRect);
```

Note that this strategy will result in the picture-in-picture movie being distorted, unless its original dimensions are proportional to the dimensions of the main movie. It would be easy, however, to modify this code to retain the original proportions of the picture-in-picture movie while limiting its largest dimension to one-third the height or width of the main movie. (This refinement is left as an exercise for the reader.) Finally, let's keep track of the picture-in-picture movie and its new rectangle in the application data structure attached to the main movie window.

```
(**myAppData).fPIPRect = myRect;
(**myAppData).fPIPMovie = myPIPMovie;
```

Setting the Main Movie's Display Clipping Region

If we simply started playing the picture-in-picture movie and the main movie using the geometry we've set up so far, we would get an annoying flickering caused by the main movie image being drawn over the existing picture-in-picture movie image, which in turn is then refreshed over the main movie image. To avoid this, we want to set up the main movie so that it doesn't draw within the rectangle that encloses the picture-in-picture movie. We can do this by defining a new display clipping region for the main movie.

A movie's *display clipping region* is the portion of the movie box in which the movie image is to be drawn. That is to say, the display clipping region defines a clipping region that is applied to a movie just before it is displayed. (A movie also has a clipping region, but that region is applied much earlier in the imaging pipeline and isn't relevant to us here.) By default, a movie's display clipping region exactly coincides with its movie box, but it's possible to set any subregion of the movie box as the display clipping region. In the current situation, we want the main movie's display clipping region to be the rectangle occupied by the main movie minus the rectangle occupied by the picture-in-picture movie.

To calculate this clipping region, we can use standard QuickDraw functions to get the regions corresponding to the two movie boxes and then subtract one region from the other. We set this region as the main movie's display clipping region using the SetMovieDisplayClipRgn function. This is all accomplished in the function QTMT_SetMovieClipRegion defined in Listing 14. The Movie Toolbox copies the display clipping region that we pass it, so we can safely dispose of that region, as well as the two intermediate regions we use to calculate the display clipping region, before exiting QTMT_SetMovieClipRegion.

Listing 14: Setting the display clipping region of the main movie.

QTMT_SetMovieClipRegion

```
OSErr QTMT_SetMovieClipRegion (WindowObject theWindowObject)
{
  Rect                  myRect;
  RgnHandle             myMovieRegion;
  RgnHandle             myPIPRegion;
  RgnHandle             myClipRegion;
  ApplicationDataHdl    myAppData = NULL;
```


Commit this to "Memory!"

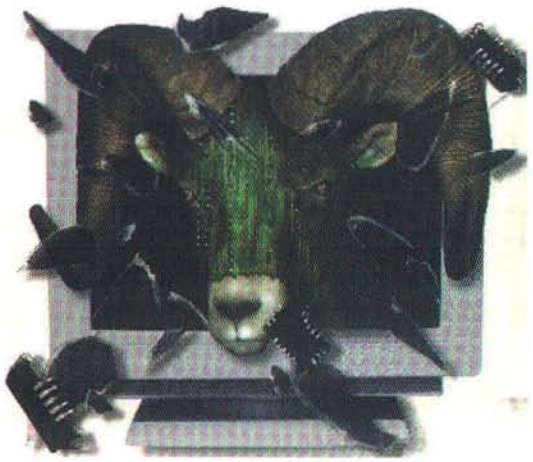
Cisco

Sun

HP

Compaq

Apple



SGI

Dell

IBM

Gateway

And many more...

Rocky Mountain Ram is a manufacturer of the highest quality computer memory. Upgrades for all major brands of Routers, PC's, Apple, Workstations, Servers, Laptops and Printers. Rocky Mountain Ram maintains extensive inventory for fast service and factory direct pricing. All design, acquisition and manufacturing are in-house. Rocky Mountain Ram has over 60 years of combined experience in the computer memory industry, and specializes in Government, Internet and backbone providers, University and Fortune 500 companies. Rocky Mountain Ram is dedicated to meeting our customers memory requirements in a responsive, knowledgeable, and supportive manner.

Visit our website at www.ram-it.com (800) 543-0932 Fax (303) 413-8255

```
  // get the application-specific data
  myAppData =
(ApplicationDataHdl)QTFrame_GetAppDataFromWindowObject
                                          (theWindowObject);
  if (myAppData == NULL)
    return(paramErr);

  myMovieRegion = NewRgn();
  myPIPRegion = NewRgn();
  myClipRegion = NewRgn();

  GetMovieBox((**theWindowObject).fMovie, &myRect);
  RectRgn(myMovieRegion, &myRect);

  myRect = (**myAppData).fPIPRect;
  if (!EmptyRect(&myRect))
    MacInsetRect(&myRect, -kPIPBorderSize, -kPIPBorderSize);
  RectRgn(myPIPRegion, &myRect);

  DiffRgn(myMovieRegion, myPIPRegion, myClipRegion);

  SetMovieDisplayClipRgn((**theWindowObject).fMovie,
                                          myClipRegion);

  DisposeRgn(myMovieRegion);
  DisposeRgn(myPIPRegion);
  DisposeRgn(myClipRegion);

  return(noErr);
}
```

Notice that we call MacInsetRect inside of QTMT_SetMovieClipRegion to expand the picture-in-picture movie box by a preset amount (kPIPBorderSize) before converting that box to a region. We do this to provide a few pixels of space in which to draw a border around the picture-in-picture movie, which gives it a somewhat better appearance. (See Listing 15 below for the code that draws the border.)

There is one final thing to keep in mind: if the main movie were being played using a movie controller component, then we should set the movie display clipping region using the function MCSetClip instead of SetMovieDisplayClipRgn.

Playing the Picture-In-Picture Movie

Now that we've opened the picture-in-picture movie and positioned it within the main movie window, we need to start the picture-in-picture movie playing and keep it playing. All the lessons we learned earlier in "Preparing Movies For Playback" apply here as well, so we'll kick things off by preprerolling the picture-in-picture movie, like this:

```
PrePrerollMovie(myPIPMovie, 0,
GetMoviePreferredRate(myPIPMovie), gMoviePPRollCompleteProc,
(void *)theWindowObject);
```

We're using the same movie preprerolling completion function that we used for the main movie, which prerolls the movie as soon as the preprerolling is completed. But you'll recall from Listing 7 that the completion procedure includes a few extra lines that apply only to the picture-in-picture movie:

```
if (theMovie == (**myAppData).fPIPMovie) {
  QTMT_SetMovieLoopState(theMovie, kNormalLooping);
  StartMovie(theMovie);
}
```

Since we're not going to provide a way for the user to start or stop the picture-in-picture movie, we call QTMT_SetMovieLoopState (defined in Listing 11) to set the movie looping state to normal looping. The movie will then keep playing, over and over, until we close the window or replace it with some other picture-in-picture movie. Then, we just call StartMovie to start the picture-in-picture movie playing.

So the only thing that remains for us to do is task the movie periodically to keep it playing back smoothly. As before, this is something we want to do in our idle routine (QTApp_Idle), where we are already calling MoviesTask on the main movie. We have several options here. First, we could just add these few lines of code to QTApp_Idle, to explicitly task the picture-in-picture movie:

```
if ((**myAppData).fPIPMovie != NULL)
  MoviesTask((**myAppData).fPIPMovie, DoTheRightThing);
```

Or, we could pass the value NULL as the first parameter to MoviesTask, like this:

```
MoviesTask(NULL, DoTheRightThing);
```

When we specify NULL in place of a movie, we're instructing the Movie Toolbox to service *all* open movies belonging to the application. Remember, however, that we want to draw a border around the picture-in-picture movie. So the option we'll actually choose will be to call an application-defined function:

```
if ((**myAppData).fPIPMovie != NULL)
  QTMT_DrawPictureInPictureMovie(myWindowObject);
```

QTMT_DrawPictureInPictureMovie simply calls MoviesTask on the picture-in-picture movie and draws a nice-looking, pseudo-grayscale border around the movie, as shown in Listing 15.

Listing 15: Drawing the picture-in-picture movie and its border.

QTMT_DrawPictureInPictureMovie

```
OSErr QTMT_DrawPictureInPictureMovie
                              (WindowObject theWindowObject)
{
  ApplicationDataHdl  myAppData = NULL;

  if (theWindowObject == NULL)
    return(paramErr);

  myAppData =
(ApplicationDataHdl)QTFrame_GetAppDataFromWindowObject
                                          (theWindowObject);
  if (myAppData == NULL)
    return(paramErr);

  // draw the current frame of the picture-in-picture movie
  if ((**myAppData).fPIPMovie != NULL)
    MoviesTask((**myAppData).fPIPMovie, DoTheRightThing);

  if (!EmptyRect(&(**myAppData).fPIPRect)) {
    PenState      myPenState;
    Rect          myRect;

    myRect = (**myAppData).fPIPRect;

    // draw a pseudo-grayscale border
    GetPenState(&myPenState);

    MacInsetRect(&(**myAppData).fPIPRect, -kPIPBorderSize, -
                                          kPIPBorderSize);
    RGBForeColor(&gBlack);
    MacFrameRect(&(**myAppData).fPIPRect);
    MacInsetRect(&(**myAppData).fPIPRect, 1, 1);
    RGBForeColor(&gWhite);
    MacFrameRect(&(**myAppData).fPIPRect);
    MacInsetRect(&(**myAppData).fPIPRect, 1, 1);
```

```
    RGBForeColor(&gLGray);
    MacFrameRect(&(**myAppData).fPIPRect);

    (**myAppData).fPIPRect = myRect;

    SetPenState(&myPenState);
  }
  return(noErr);
}
```

Moving the Picture-In-Picture Movie

Let's finish up by adding a simple enhancement to our picture-in-picture movie playback capability: whenever the user clicks the mouse button when the cursor is inside of the main movie window, let's move the picture-in-picture movie so that its upper-left corner is at the location of the click. Our basic cross-platform QuickTime framework passes mouse button clicks to the application function QTApp_HandleContentClick whenever those clicks fall within the window's content area. So all we need to do is add some code to QTApp_HandleContentClick.

Moving the picture-in-picture movie rectangle is very simple. First, we need to convert the point that's passed to us in the event record into local coordinates:

```
myPoint = theEvent->where;
GlobalToLocal(&myPoint);
```

Then we need to calculate the new movie box for the picture-in-picture movie. The size of the movie box isn't changing, only its upper-left corner. So we can call MacOffsetRect to figure out where to place the movie box and SetMovieBox to actually move it.

```
MacOffsetRect(&(**myAppData).fPIPRect,
      myPoint.h - (**myAppData).fPIPRect.left,
      myPoint.v - (**myAppData).fPIPRect.top);

SetMovieBox((**myAppData).fPIPMovie,
&(**myAppData).fPIPRect);
```

Finally, we need to recalculate the display clipping region of the main movie, like this:

```
QTMT_SetMovieClipRegion(myWindowObject);
```

The complete procedure for moving the picture-in-picture movie in response to mouse clicks is shown in Listing 16.

Listing 16: Moving the picture-in-picture movie within the main movie.

QTApp_HandleContentClick

```
void QTApp_HandleContentClick (WindowReference theWindow,
EventRecord *theEvent)
{
  CGrafPtr              mySavedPort;
  WindowObject          myWindowObject = NULL;
  ApplicationDataHdl    myAppData = NULL;

  GetPort(&mySavedPort);
  MacSetPort(QTFrame_GetPortFromWindowReference(theWindow));

  myWindowObject =
QTFrame_GetWindowObjectFromWindow(theWindow);
  myAppData =
(ApplicationDataHdl)QTFrame_GetAppDataFromWindow(theWindow);
  if ((myWindowObject == NULL) || (myAppData == NULL))
    return;

  // move the picture-in-picture movie to the specified location
  if ((**myAppData).fPIPMovie != NULL) {
    Point           myPoint;

    myPoint = theEvent->where;
    GlobalToLocal(&myPoint);

    // if the main movie isn't playing, erase the rectangle currently occupied
    // by the picture-in-picture movie
    if (!QTMT_IsMoviePlaying((**myWindowObject).fMovie)) {
      Rect      myRect;

      myRect = (**myAppData).fPIPRect;
      MacInsetRect(&myRect, kPIPBorderSize,
                                          kPIPBorderSize);
      EraseRect(&myRect);
    }

    MacOffsetRect(&(**myAppData).fPIPRect,
          myPoint.h - (**myAppData).fPIPRect.left,
          myPoint.v - (**myAppData).fPIPRect.top);

    SetMovieBox((**myAppData).fPIPMovie,
                                  &(**myAppData).fPIPRect);

    QTMT_SetMovieClipRegion(myWindowObject);
  }

  MacSetPort(mySavedPort);
}
```

Conclusion

Keep in mind that the point of this article is not that you should be playing movies using the Movie Toolbox alone, without the assistance of movie controllers. Indeed, as we've seen, movie controllers do lots and lots of useful work for us, as well as a good deal of humdrum housekeeping (such as setting the movie rate to 0 when the movie reaches the end). And movie controllers have a certain measure of built-in "future compatibility", as evidenced by the fact that movie controllers now prepreroll movies automatically. This means that, if you had written a movie-playing application several years ago that used movie controllers, you wouldn't have had to change it much, if at all, to support the new streaming movies introduced in QuickTime 4.

Still, there are occasions when you might need to manage a movie without using a movie controller, and we've seen (fairly exhaustively, I hope) what you need to do to accomplish that. But, our larger goal here was to get acquainted with the Movie Toolbox itself, to get a taste of its capabilities and to see how to operate on movies directly. As we steadily progress in our journey through QuickTime, we'll come to depend more and more on the Movie Toolbox and less and less on movie controllers.

Credits

Thanks are due to ***Sam Busbell*** and ***Peter Hoddie*** for clarifying several issues about the Movie Toolbox for me. And, thanks to ***Eric Blanpied*** for providing the rendered toy train movie shown in **Figure 3**. MT

PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

Sum of Powers

A year or so back, Ken Slezak wrote me to say: "Recently my 7th grade son showed me an extra credit math problem. Given a number from 1 to 100, create that number from one, two, or three squared numbers added or subtracted together.... It turned out to be more difficult than I thought! Might this make an interesting programming challenge?" Well, Ken, we're about to find out.

First, though, we need to spice up the problem a bit. I've been trying to make the problems a little easier of late, but the 7th grade level, even the 7th grade extra credit level, would be a bit too easy. First, we'll expand the problem domain to include any positive number that fits into a signed 32-bit long integer. Second, we'll expand the number of terms allowed. And third, instead of limiting the problem to only squared numbers, we'll allow any positive exponent greater than 1.

The prototype for the code you should write is:

```
typedef struct IntegerPower {
  long value;   /* term is sign*value^power */
  short power;  /* power = 2,3,4,... */
  short sign;   /* +1 or -1 */
} IntegerPower;

long /* number of factors */ SumOfPowers (
  long result,              /* terms need to sum to this result */
  IntegerPower terms[],     /* return terms sign*value^power here */
  long maxTerms             /* maximum number of terms allowed */
);
```

Your SumOfPowers routine will be called a number of times. Each time, you must identify a set of terms which sum to the specified result. Each term is a value raised to an integer power greater than 1, multiplied by a sign value. You should return the number of terms used to form the result, or zero if the result cannot be formed with maxTerms terms.

The winner will be the solution that forms the desired results with the minimum number of terms. A time penalty of one term will be added per 100 milliseconds of execution time. All solutions must be calculated at execution time; any entry that precalculates a solution will be disqualified.

This will be a native PowerPC Challenge, using the CodeWarrior Pro 5 environment. Solutions may be coded in C, C++, or Pascal. Solutions in Java will also be accepted, but Java entries must be accompanied by a test driver that uses the interface provided in the problem statement.

Ken Slezak earns 2 Challenge points for suggesting this month's Challenge.

Three Months Ago Winner

Congratulations to **Jonathan Taylor (Blandford, Dorset, U.K.)** for winning the December, 1999, Costas Array Challenge. The Costas Challenge required contestants to produce all of the arrays of a given dimension that satisfied two criteria. First, each row and each column of the array must have exactly one "1", with the remaining entries being zeros. Second, the line connecting a pair of "1"s in an array may not have the same slope as the line connecting any other pair of "1"s. Costas arrays have properties that make it an ideal discrete waveform for some sensor applications. The number of Costas arrays of order N increases until N==17, after which it decreases at least until N==23. This Challenge attracted 16 entries, 13 of which worked correctly.

Jonathan's solution is recursive for array sizes greater than 17, but he gains speed by unrolling the code for smaller orders. He also took advantage of the fact that assembly language was permitted for this Challenge. For these smaller arrays, he maintained the position of the "1"s in each row in separate registers, eliminating a significant number of memory accesses. This optimization allowed Jonathan's entry to run approximately 15% faster than the second place entry by Ernst Munter. Jonathan included a nice description of the recursive and unrolled algorithms in the preface to his code, so I won't repeat his description here.

I evaluated every entry by having them calculate the Costas arrays of orders 5 through 13, inclusive. I then extended the tests to include arrays of orders through 15. Finally, I tested the top entries with arrays of orders through 17. In evaluating the correctness of the arrays generated, I made use of some code provided by Ludovic Nicolle. Ludo's code verified that each array produced met the criteria for a Costas array, and that each array was distinct from any other array. Solutions were judged correct if the number of valid, unique Costas arrays produced equaled the known number of Costas arrays of a given order. Judging correctness would have been more difficult if any of the solutions had completed problems where the number of Costas arrays is unknown; fortunately (or unfortunately, depending on your point of view), none of the entries completed problems of that size. Several contestants observed that execution time increased by a factor of ~5.3 from one order to the next, and some estimated that 1000 fast PowerMacs should be able to calculate the arrays of order 24 in a couple of weeks. Perhaps we should turn this over to the distributed.net folks.

The table below lists, for each of the solutions submitted, the cumulative time required to calculate the Costas arrays up to orders 13, 15, and 17. It also indicates the code size, data size, and programming language used by each solution. As usual, the number in parentheses after the entrant's name is the total number of Challenge points earned in all Challenges prior to this one.

Name	Time (13) (secs)	Time (15) (secs)	Time (17) (secs)	Errors	Code Size	Data Size	Lang
Jonathan Taylor (4)	7.69	228.79	1246.12	0	21334	200	C/asm
Ernst Munter (547)	9.31	270.32	1477.03	0	4668	8608	C++/asm
Ludovic Nicolle (48)	10.02	289.08	N/A	0	2612	24	C++
Xan Gregg (116)	12.06	340.65	N/A	0	4428	8	C++
Rob Shearer (41)	12.06	346.45	N/A	0	158720	10300	C++
Willeke Rieken (61)	19.36	540.05	N/A	0	2420	8	C
Tom Saxton (158)	33.16	1011.24	N/A	0	1500	8	C
Greg Sadetsky (2)	40.74	1122.02	N/A	0	2044	8	C
Michael Lewis	51.94	1592.36	N/A	0	1528	72	C
Randy Boring (112)	57.32	1801.37	N/A	0	1516	24	C++
Sebastian Maurer (77)	59.83	1927.93	N/A	0	1216	74	C
Brian Hall	125.35	3642.11	N/A	0	2880	412	C
Brady Duga (10)	130.50	N/A	N/A	0	4152	371	C++
D. L.	N/A	N/A	N/A	1			
C. L.	N/A	N/A	N/A	1			
B. B.	N/A	N/A	N/A	1			

Top Contestants

Listed here are the Top Contestants for the Programmer's Challenge, including everyone who has accumulated 10 or more points during the past two years. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	247	10.	Jones, Dennis	12
2.	Saxton, Tom	139	11.	Hart, Alan	11
3.	Maurer, Sebastian	67	12.	Duga, Brady	10
4.	Rieken, Willeke	51	13.	Hewett, Kevin	10
5.	Heithcock, JG	43	14.	Murphy, ACC	10
6.	Shearer, Rob	43	15.	Selengut, Jared	10
7.	Boring, Randy	39	16.	Strout, Joe	10
8.	Taylor, Jonathan	24	17.	Varilly, Patrick	10
9.	Brown, Pat	20			

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points
2nd place	10 points
3rd place	7 points
4th place	4 points
5th place	2 points
finding bug	2 points
suggesting Challenge	2 points

Here is Jonthan Taylor's winning Costas solution:

Costas.c
Copyright © 1999 Jonathan Taylor

```
/*
```
The algorithm works by attempting to find a position for a '1' in the first row of the array, and then to work down the array attempting to fill each row in turn until there is no permissible move, in which case the program backtracks to the previous row and tries the next permissible position. At the beginning of each row, it calculates a bit-mask which shows which positions have been forbidden by the choice of position on the previous rows.

There are two conditions that are detected:
1. A '1' is already in the column
2. Putting a '1' in this position would create three in a row:

Let a A and B be the x position of the '1' in two previous rows. Every existing combination of A and B is checked in turn. C is another row's x position. This row is the one which, when combined with B and C will cause one of the positions in the current row to be forbidden. For example:

```
1000 0000 row described by A
0000 0010
0010 0000 row described by B
0100 0000 row described by C
0001 0000
```

(the next row is the one currently being determined)
the '1' in row B is two right and two down of the one in A, so the position two right and two down from the '1' in row C is forbidden. The program will record this information appropriately

The program was a recursive function which stores the position of the 1 in each row (rowPos) in RAM, as well as the bitField for each row. The bitField is a long where each bit represents one of the positions that the '1' could be placed in the current row. If a bit is set, that position is forbidden by the rules as applied to the rows that have already been decided. Unused bits are also one. Note that each level of recursion has its own bitField variable, each of which will be very different.

In each recursion, it sets bitField to the appropriate value. If bitField is now equal to
-1, there are no permissible places to put the '1' on this row, and the function returns to the previous recursion. If it does not return, it then checks each bit in bitField to see if it is a zero. If it is, that is a permissible place for the '1'. It records this selection and calls the next level of recursion (when that level returns, execution will resume by trying the next bit in bitField).

If by placing a '1' at this level the array has been successfully filled, the array has the correct values entered in it, and the function then returns to the previous level of recursion in order to continue the search for the next solution. It should be noted that if an array is a valid solution, then its horizontal reflection is also a different, valid solution (for 2 or more rows). That is also recorded in the solution list. This allows the running time to be roughly halved, because once the first row has been half traversed, all the solutions will have been found.

The problem with that is that the program must be stopped halfway through the arrays in order to prevent there being repeats. The condition is slightly different for even- and odd-sized arrays, so one of two different tests is used depending on the array size.

This version was used for cases of n>17. Version 2 handles smaller cases (much faster!). The underlying algorithm is the same, though. It could easily be extended to higher ones by writing more code.

Version 2

I suspected that the limiting factor for the speed of the previous example was probably memory accesses. In calculating bitField, the positions of the '1's in previous rows are used over and over again. However, they are not used in order, and neither I nor the complier could find any effective way of caching the values in registers to save on RAM accesses. However, I realised there were in fact enough registers to fit them all into registers.

This strategy is extremely fast, as it requires only one write to RAM when calling the next recursion level, and one read when returning. However there is a problem. I have been unable to discover a way of indirectly addressing a register. It is not possible to design a loop that will access,

say, row_position[n] if it is stores in register n. It was therefore necessary to unroll all the loops completely (If anyone wants to tell me how I could have performed indirect addressing of registers, I'd love to know!).

Since the loops were of variable length depending on the level of recursion, it was necessary to write new code for every level of recursion, with 'goto's to "call" the next level of recursion. This has the huge drawback that new code must be written in order to allow the program do determine larger arrays. The competition code was able to deal with up to 17x17 arrays. What I should have done was to write a program to generate the C code, since the unrolling is completely deterministic, although fairly complex. As it was, I had to spend a long time checking for errors in the unrolling.

By the 17th row, there are 140 calls of SetBit()! There are only 17 bits to be set. This is a huge overkill if it turns out there is no place to go anyway! As a result, I inserted the line:

```
if(tempBitField==-1) goto endSetForbiddenXX;
```

after every 5 SetBit() s. This allows the subsequent calls to be skipped if there is already no place left to go. I have no idea how often they should be put - I didn't have time to experiment - and they take up valuable execution time. With some serious experimentation, a better distribution of the 'if's could almost certainly be found.

This version has several limitations. The problem of loop unrolling limiting the array size can be partly solved by producing an automatic C code generator as described. The other potential problem is lack of registers. I reckon about 8-10 are needed in addition to the row-position ones. With current processors having 32 integer registers, 22x22 arrays could potentially be calculated. After that, the values would have to spill over into RAM, causing some, but not initially much, slowdown.

*/

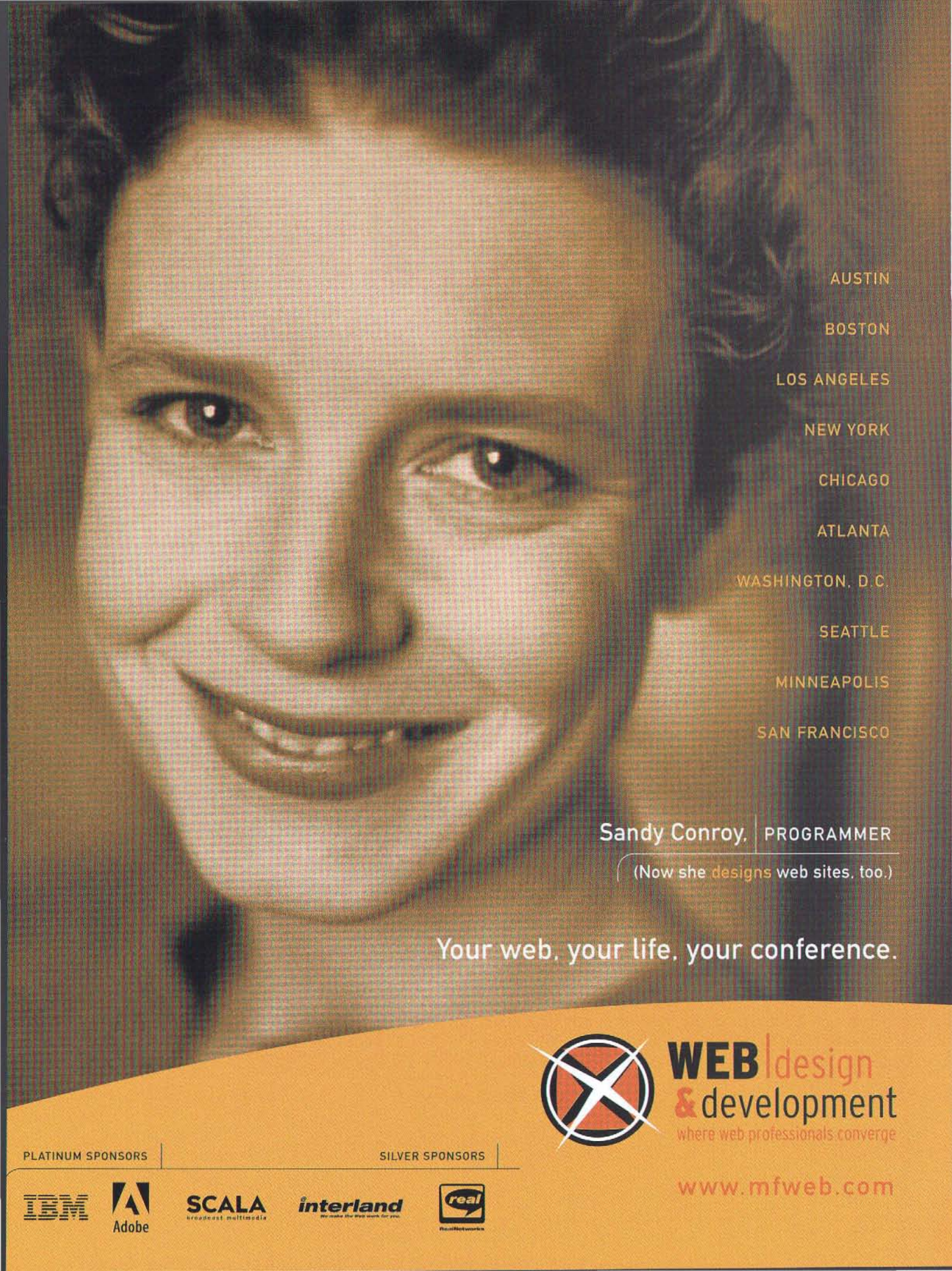
```
#include "Costas.h"
#include <MacTypes.h>

// this macro is the assembly code for the detection of a three-in-a-row
// situation
#define SetBit(A,B,C)  add          r0,A,B;
\
                                subf        r0,C,r0;
  \
                                slw
r0,kOne,r0;   \
                                or
tempBitField,tempBitField,r0

// this macro is the assembly code for detection of an already-filled
// column
#define HadBit(A)     slw     r0,kOne,A;        \
              or      tempBitField,tempBitField,r0

#pragma optimization_level 2
/* I don't remember exactly why this was needed. If the level is zero, the
'register' keywords are ignored. If it is too high, maybe it just took for
too long to compile without gaining anything... */

void DoCostasRow(long row);
long EnumerateCostasInRegisters(int n,unsigned long
*costasArrays);

long      gCount,gMax_Rows;
unsigned long *gCostasArrays;

long EnumerateCostas(int n,unsigned long
*costasArrays)
{
  if(n>17)
  {
    gMax_Rows = n;
    gCostasArrays=costasArrays;
    DoCostasRow(0);          //call recursive function
    return(gCount);
  }
  else

return(EnumerateCostasInRegisters(n,costasArrays));
}

void DoCostasRow(long row)
{
  register long     n,m,bitField,bitNum;
  static long       doExit; //used to signal when all arrays found
  static long       count;      //number found
  static long       max_rows;   //dimensions of array
  static long       blankBitField;
  static long       rowPosStore[32]; //x position of the '1'
                                     //in each row
  static unsigned long *costasArrays;

  if(row == 0)    //setup code
  {
    count = 0;
    max_rows = gMax_Rows;
    blankBitField = (-1)^((1<<max_rows)-1);
    costasArrays=gCostasArrays;
    doExit=false;
  }
  // This next bit detects which positions are forbidden
  // See introduction for more details
  bitField = blankBitField; //make all x positions permissible
  for(n=0;n<row;n++)
  {
    bitField|=(1<<rowPosStore[n]); //forbid (condition #1)
    for(m=0;m<n;m++)                //forbid (condition #2)
      bitField |= (1<<
      (rowPosStore[n]+rowPosStore[row-1-m]-
rowPosStore[n-m-1]));
  }
  if(bitField == -1)
    return;           //all positions are forbidden

  for(bitNum=0;bitNum<max_rows;bitNum++)
//try each position in turn
  {
    if((bitField&(1<<bitNum)) == 0)    //permissible
    {
      //Next comes code for checking when we are done
      //(see introduction)
      if(max_rows%2 == 1)  //odd-sized array
      {
        if(row==1)
        {
          if(bitNum>max_rows/2 &&
rowPosStore[0]==max_rows/2)
          {
            gCount=count;
            doExit=true;
            return;
          }
        }
      }
      else if(row==0)  //even
      {
        if(bitNum>=max_rows/2)
        {
          gCount=count;
          return;
        }
      }

      rowPosStore[row] = bitNum;
      if(row == max_rows-1)       //got a complete array
      {
        for(n=0;n<max_rows;n++)     //record it
          costasArrays[count*max_rows+n] =
(1<<rowPosStore[n]);
        count++;
        if(max_rows != 1)
        {
          for(n=0;n<max_rows;n++)
                //and its horizontal reflection
            costasArrays[count*max_rows+n] =
              ((1<<(max_rows-1))>>rowPosStore[n]);
          count++;
        }
        else          //array is 1x1 - got the solution so exit
          gCount=count;
        return;
      }
```

```
      DoCostasRow(row+1);   //call next level of recursion
      if(doExit) return;    //is set when all arrays are found
    }
  }
}

#define STORE0 RP00=bitNum
#define STORE1 RP01=bitNum
#define STORE2 RP02=bitNum
#define STORE3 RP03=bitNum
#define STORE4 RP04=bitNum
#define STORE5 RP05=bitNum
#define STORE6 RP06=bitNum
#define STORE7 RP07=bitNum
#define STORE8 RP08=bitNum
#define STORE9 RP09=bitNum
#define STORE10  RP10=bitNum
#define STORE11  RP11=bitNum
#define STORE12  RP12=bitNum
#define STORE13  RP13=bitNum
#define STORE14  RP14=bitNum
#define STORE15  RP15=bitNum

#define LOAD0 bitNum=RP00
#define LOAD1 bitNum=RP01
#define LOAD2 bitNum=RP02
#define LOAD3 bitNum=RP03
#define LOAD4 bitNum=RP04
#define LOAD5 bitNum=RP05
#define LOAD6 bitNum=RP06
#define LOAD7 bitNum=RP07
#define LOAD8 bitNum=RP08
#define LOAD9 bitNum=RP09
#define LOAD10  bitNum=RP10
#define LOAD11  bitNum=RP11
#define LOAD12  bitNum=RP12
#define LOAD13  bitNum=RP13
#define LOAD14  bitNum=RP14
#define LOAD15  bitNum=RP15

long EnumerateCostasInRegisters(int n,
                  unsigned long *costasArrays)
{
  register long   bitField;
                  /* this is a set of bits that records which
                     positions are forbidden due to the choices
                       for previous rows. A '1' means the position
                         is forbidden */
  register long tempBitField;
      /* when a new level of recursion is begun, this
         stores the new value of 'bitField' as it is
           calculated. But if there are no legal positions,
             an immediate return occurs. bitField is only
               updated if there is al least one position, since
                 the bitField value for the previous recursion
                   level must be stored in RAM - a slow process! */
  register long bitNum;
          //the current x position in the row under consideration
  register long   blankBitField;
  register long   kOne;
          //code is faster if this is a register instead of a constant
  register long
RP00,RP01,RP02,RP03,RP04,RP05,RP06,RP07,

RP08,RP09,RP10,RP11,RP12,RP13,RP14,RP15;
    //RPn ('row position n') stores the x coordinate of the '1' on row n
  long        bitFieldStore[32];
  /* these store the bitFields as calculated for previous recursion levels
  They must be reloaded after a return to a previous recursion level */
  long        count;                //number of solutions found

  kOne=1;
  count=0;
begin0:
  blankBitField=(-1)^((kOne<<n)-1);
  bitField=blankBitField;
  bitFieldStore[0]=bitField;

  for(bitNum=0;bitNum<n;)   //try every x position in turn
  {
    if((bitField&(kOne<<bitNum))==0)
                            //if this position not forbidden
    {
      //RowPosStore[row]=bitNum;
      STORE0;

      if(n==1)              //record layout!
      {
        costasArrays[count*n]=(1<<RP00);
                            //record in result array
        count++;
        return count; //return number found
      }
      goto begin1;      //try the next row...
    }
returnPoint0:
    bitNum++;
  }
doReturn0:
  return count;

begin1:
  tempBitField=blankBitField;
  asm { HadBit(RP00) } //can't be below the '1' in the first row
  bitField=tempBitField;
  bitFieldStore[1]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE1;

      if(n==2)          //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        count++;
        costasArrays[count*n]=(2>>RP00);
        costasArrays[count*n+1]=(2>>RP01);
        count++;
        goto doReturn1;
      }
      goto begin2;
    }
returnPoint1:
    bitNum++;
  }
doReturn1:
  // bitNum=RowPosStore[row];
  LOAD0;
  bitField=bitFieldStore[0];
  goto returnPoint0;

begin2:
  tempBitField=blankBitField;
setForbidden2:
   asm{   HadBit(RP00) } //set forbidden bits (condition #1)
   asm{   HadBit(RP01) }

   asm{   SetBit(RP01,RP01,RP00)}
      //set forbidden bits (condition #2)
endSetForbidden2:
  if(tempBitField==-1)
      //give up now if no positions permitted
    goto returnPoint1;
  bitField=tempBitField;
  bitFieldStore[2]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE2;

      if(n==3)          //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
```

```
        count++;
        costasArrays[count*n]=(4>>RP00);
        costasArrays[count*n+1]=(4>>RP01);
        costasArrays[count*n+2]=(4>>RP02);
        count++;
        goto doReturn2;
      }
      goto begin3;
    }
returnPoint2:
    bitNum++;
  }
doReturn2:
  // bitNum=RowPosStore[row];
  LOAD1;
  bitField=bitFieldStore[1];
  goto returnPoint1;

begin3:
  tempBitField=blankBitField;
setForbidden3:
   asm{ HadBit(RP00)}
   asm{ HadBit(RP01)}
   asm{ HadBit(RP02)}

   asm{         SetBit(RP02,RP01,RP00) }
   asm{         SetBit(RP02,RP02,RP01) }
   asm{         SetBit(RP01,RP02,RP00) }

endSetForbidden3:
  if(tempBitField==-1)
    goto returnPoint2;
  bitField=tempBitField;
  bitFieldStore[3]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE3;
      if(n==4)       //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        count++;
        costasArrays[count*n]=(8>>RP00);
        costasArrays[count*n+1]=(8>>RP01);
        costasArrays[count*n+2]=(8>>RP02);
        costasArrays[count*n+3]=(8>>RP03);
        count++;
        goto doReturn3;
      }
      goto begin4;
    }
returnPoint3:
    bitNum++;
  }
doReturn3:
  // bitNum=RowPosStore[row];
  LOAD2;
  bitField=bitFieldStore[2];
  goto returnPoint2;

begin4:
  tempBitField=blankBitField;
setForbidden4:
   asm{ HadBit(RP00)
      HadBit(RP01)
      HadBit(RP02)
      HadBit(RP03)

      SetBit(RP03,RP01,RP00)
      SetBit(RP03,RP02,RP01)
      SetBit(RP03,RP03,RP02)
      SetBit(RP02,RP02,RP00)
      SetBit(RP02,RP03,RP01)
      SetBit(RP01,RP03,RP00)    }

endSetForbidden4:
  if(tempBitField==-1)
    goto returnPoint3;
  bitField=tempBitField;
  bitFieldStore[4]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE4;
      if(n==5)       //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        count++;
        costasArrays[count*n]=(16>>RP00);
        costasArrays[count*n+1]=(16>>RP01);
        costasArrays[count*n+2]=(16>>RP02);
        costasArrays[count*n+3]=(16>>RP03);
        costasArrays[count*n+4]=(16>>RP04);
        count++;
        goto doReturn4;
      }
      goto begin5;
    }
returnPoint4:
    bitNum++;
  }
doReturn4:
  // bitNum=RowPosStore[row];
  LOAD3;
  bitField=bitFieldStore[3];
  goto returnPoint3;

begin5:
  tempBitField=blankBitField;
setForbidden5:
   asm {  HadBit(RP00)
      HadBit(RP01)
      HadBit(RP02)
      HadBit(RP03)
      HadBit(RP04)

      SetBit(RP04,RP01,RP00)
      SetBit(RP04,RP02,RP01)
      SetBit(RP04,RP03,RP02)
      SetBit(RP04,RP04,RP03)
      SetBit(RP03,RP02,RP00)  }
```

/*In most cases for large row numbers, I suspect that all positions will be very quickly seen to be forbidden. All the subsequent SetBit() calls are unnecessary in that instance. This check allows them to be skipped out. It appears after every five SetBit() calls (see introduction for comments) */

```
  if(tempBitField==-1)
    goto endSetForbidden5;
  asm {
      SetBit(RP03,RP03,RP01)
      SetBit(RP03,RP04,RP02)
      SetBit(RP02,RP03,RP00)
      SetBit(RP02,RP04,RP01)
      SetBit(RP01,RP04,RP00)    }

endSetForbidden5:
  if(tempBitField==-1)
    goto returnPoint4;
  bitField=tempBitField;
  bitFieldStore[5]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE5;
```


AUSTIN
BOSTON
LOS ANGELES
NEW YORK
CHICAGO
ATLANTA
WASHINGTON, D.C.
SEATTLE
MINNEAPOLIS
SAN FRANCISCO
Sandy Conroy, | PROGRAMMER
(Now she designs web sites, too.)
Your web, your life, your conference.
WEB design & development
where web professionals converge
www.mfweb.com
PLATINUM SPONSORS
SILVER SPONSORS
IBM
Adobe
SCALA
broadcast multimedia
interland
real
RealNetworks

```
      if(n==6)        //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        count++;
        costasArrays[count*n]=(32>>RP00);
        costasArrays[count*n+1]=(32>>RP01);
        costasArrays[count*n+2]=(32>>RP02);
        costasArrays[count*n+3]=(32>>RP03);
        costasArrays[count*n+4]=(32>>RP04);
        costasArrays[count*n+5]=(32>>RP05);
        count++;
        goto doReturn5;
      }
      goto begin6;
    }
returnPoint5:
    bitNum++;
  }
doReturn5:
  // bitNum=RowPosStore[row];
  LOAD4;
  bitField=bitFieldStore[4];
  goto returnPoint4;
```

/* the code then continues with begin6 up to begin16. The pattern the code is following after each 'begin' should be clear now!

The size of the code grows with n^2, and becomes completely dominated by SetBit(). This is definitely the place for further optimization! Several things spring to mind.
At the moment I suspect the SetBit() macros are executed in strict order, since the tempBitField register is always being modified. By using a number of different registers that are written to, it might be possible to allow a much greater degree of instruction interleaving, which looks as if it could at least double the speed if done cleverly?

The other place for improvement is the the 'if(tempBitField==-1)' lines (see note in introduction).
*/

```
begin6:
  tempBitField=blankBitField;
setForbidden6:
  asm { HadBit(RP00)
        HadBit(RP01)
        HadBit(RP02)
        HadBit(RP03)
        HadBit(RP04)
        HadBit(RP05)

        SetBit(RP05,RP01,RP00)
        SetBit(RP05,RP02,RP01)
        SetBit(RP05,RP03,RP02)
        SetBit(RP05,RP04,RP03)
        SetBit(RP05,RP05,RP04) }
    if(tempBitField==-1)
      goto endSetForbidden6;
  asm { SetBit(RP04,RP02,RP00)
        SetBit(RP04,RP03,RP01)
        SetBit(RP04,RP04,RP02)
        SetBit(RP04,RP05,RP03)
        SetBit(RP03,RP03,RP00) }
    if(tempBitField==-1)
      goto endSetForbidden6;
  asm { SetBit(RP03,RP04,RP01)
        SetBit(RP03,RP05,RP02)
        SetBit(RP02,RP04,RP00)
        SetBit(RP02,RP05,RP01)
        SetBit(RP01,RP05,RP00) }

endSetForbidden6:
  if(tempBitField==-1)
    goto returnPoint5;
  bitField=tempBitField;
  bitFieldStore[6]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE6;

      if(n==7)        //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        count++;
        costasArrays[count*n]=(64>>RP00);
        costasArrays[count*n+1]=(64>>RP01);
        costasArrays[count*n+2]=(64>>RP02);
        costasArrays[count*n+3]=(64>>RP03);
        costasArrays[count*n+4]=(64>>RP04);
        costasArrays[count*n+5]=(64>>RP05);
        costasArrays[count*n+6]=(64>>RP06);
        count++;
        goto doReturn6;
      }
      goto begin7;
    }
returnPoint6:
    bitNum++;
  }
doReturn6:
  // bitNum=RowPosStore[row];
  LOAD5;
  bitField=bitFieldStore[5];
  goto returnPoint5;

begin7:
  tempBitField=blankBitField;
setForbidden7:
  asm { HadBit(RP00)
        HadBit(RP01)
        HadBit(RP02)
        HadBit(RP03)
        HadBit(RP04)
        HadBit(RP05)
        HadBit(RP06)

        SetBit(RP06,RP01,RP00)
        SetBit(RP06,RP02,RP01)
        SetBit(RP06,RP03,RP02)
        SetBit(RP06,RP04,RP03)
        SetBit(RP06,RP05,RP04) }
  if(tempBitField==-1)
    goto endSetForbidden7;
  asm { SetBit(RP06,RP06,RP05)
        SetBit(RP05,RP02,RP00)
        SetBit(RP05,RP03,RP01)
        SetBit(RP05,RP04,RP02)
        SetBit(RP05,RP05,RP03) }
  if(tempBitField==-1)
    goto endSetForbidden7;
  asm { SetBit(RP05,RP06,RP04)
        SetBit(RP04,RP03,RP00)
        SetBit(RP04,RP04,RP01)
        SetBit(RP04,RP05,RP02)
        SetBit(RP04,RP06,RP03) }
  if(tempBitField==-1)
    goto endSetForbidden7;
  asm { SetBit(RP03,RP04,RP00)
        SetBit(RP03,RP05,RP01)
        SetBit(RP03,RP06,RP02)
        SetBit(RP02,RP05,RP00)
        SetBit(RP02,RP06,RP01)
        SetBit(RP01,RP06,RP00) }

endSetForbidden7:
  if(tempBitField==-1)
    goto returnPoint6;
  bitField=tempBitField;
  bitFieldStore[7]=bitField;
```

```
  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE7:

      if(n==8)        //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        count++;
        costasArrays[count*n]=(128>>RP00);
        costasArrays[count*n+1]=(128>>RP01);
        costasArrays[count*n+2]=(128>>RP02);
        costasArrays[count*n+3]=(128>>RP03);
        costasArrays[count*n+4]=(128>>RP04);
        costasArrays[count*n+5]=(128>>RP05);
        costasArrays[count*n+6]=(128>>RP06);
        costasArrays[count*n+7]=(128>>RP07);
        count++;
        goto doReturn7;
      }
      goto begin8;
    }
returnPoint7:
    bitNum++;
  }
doReturn7:
  // bitNum=RowPosStore[row];
  LOAD6:
  bitField=bitFieldStore[6];
  goto returnPoint6;

begin8:
  tempBitField=blankBitField;
setForbidden8:
  asm { HadBit(RP00)
      HadBit(RP01)
      HadBit(RP02)
      HadBit(RP03)
      HadBit(RP04)
      HadBit(RP05)
      HadBit(RP06)
      HadBit(RP07)

      SetBit(RP07,RP01,RP00)
      SetBit(RP07,RP02,RP01)
      SetBit(RP07,RP03,RP02)
      SetBit(RP07,RP04,RP03)
      SetBit(RP07,RP05,RP04) }
  if(tempBitField==-1)
    goto endSetForbidden8;
   asm { SetBit(RP07,RP06,RP05)
      SetBit(RP07,RP07,RP06)
      SetBit(RP06,RP02,RP00)
      SetBit(RP06,RP03,RP01)
      SetBit(RP06,RP04,RP02)    }
  if(tempBitField==-1)
    goto endSetForbidden8;
  asm { SetBit(RP06,RP05,RP03)
      SetBit(RP06,RP06,RP04)
      SetBit(RP06,RP07,RP05)
      SetBit(RP05,RP03,RP00)
      SetBit(RP05,RP04,RP01)    }
  if(tempBitField==-1)
    goto endSetForbidden8;
  asm { SetBit(RP05,RP05,RP02)
      SetBit(RP05,RP06,RP03)
      SetBit(RP05,RP07,RP04)
      SetBit(RP04,RP04,RP00)
      SetBit(RP04,RP05,RP01)    }
  if(tempBitField==-1)
    goto endSetForbidden8;
  asm { SetBit(RP04,RP06,RP02)
      SetBit(RP04,RP07,RP03)
      SetBit(RP03,RP05,RP00)
      SetBit(RP03,RP06,RP01)    }
  if(tempBitField==-1)
    goto endSetForbidden8;
  asm { SetBit(RP03,RP07,RP02)
      SetBit(RP02,RP06,RP00)
      SetBit(RP02,RP07,RP01)
      SetBit(RP01,RP07,RP00)    }

endSetForbidden8:
  if(tempBitField==-1)
    goto returnPoint7;
  bitField=tempBitField;
  bitFieldStore[8]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE8:

      if(n==9)        //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        count++;
        costasArrays[count*n]=(256>>RP00);
        costasArrays[count*n+1]=(256>>RP01);
        costasArrays[count*n+2]=(256>>RP02);
        costasArrays[count*n+3]=(256>>RP03);
        costasArrays[count*n+4]=(256>>RP04);
        costasArrays[count*n+5]=(256>>RP05);
        costasArrays[count*n+6]=(256>>RP06);
        costasArrays[count*n+7]=(256>>RP07);
        costasArrays[count*n+8]=(256>>RP08);
        count++;
        goto doReturn8;
      }
      goto begin9;
    }
returnPoint8:
    bitNum++;
  }
doReturn8:
  // bitNum=RowPosStore[row];
  LOAD7:
  bitField=bitFieldStore[7];
  goto returnPoint7;

begin9:
  tempBitField=blankBitField;
setForbidden9:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm {HadBit(RP08)}

  asm { SetBit(RP08,RP01,RP00) }
  asm { SetBit(RP08,RP02,RP01) }
  asm { SetBit(RP08,RP03,RP02) }
  asm { SetBit(RP08,RP04,RP03) }
  asm { SetBit(RP08,RP05,RP04) }
    if(tempBitField==-1)
      goto endSetForbidden9;
  asm { SetBit(RP08,RP06,RP05) }
  asm { SetBit(RP08,RP07,RP06) }
  asm { SetBit(RP08,RP08,RP07) }
```

```
  asm { SetBit(RP07,RP02,RP00) }
  asm { SetBit(RP07,RP03,RP01) }
    if(tempBitField==-1)
      goto endSetForbidden9;
  asm { SetBit(RP07,RP04,RP02) }
  asm { SetBit(RP07,RP05,RP03) }
  asm { SetBit(RP07,RP06,RP04) }
  asm { SetBit(RP07,RP07,RP05) }
  asm { SetBit(RP07,RP08,RP06) }
    if(tempBitField==-1)
      goto endSetForbidden9;
  asm { SetBit(RP06,RP03,RP00) }
  asm { SetBit(RP06,RP04,RP01) }
  asm { SetBit(RP06,RP05,RP02) }
  asm { SetBit(RP06,RP06,RP03) }
  asm { SetBit(RP06,RP07,RP04) }
    if(tempBitField==-1)
      goto endSetForbidden9;
  asm { SetBit(RP06,RP08,RP05) }
  asm { SetBit(RP05,RP04,RP00) }
  asm { SetBit(RP05,RP05,RP01) }
  asm { SetBit(RP05,RP06,RP02) }
  asm { SetBit(RP05,RP07,RP03) }
    if(tempBitField==-1)
      goto endSetForbidden9;
  asm { SetBit(RP05,RP08,RP04) }
  asm { SetBit(RP04,RP05,RP00) }
  asm { SetBit(RP04,RP06,RP01) }
  asm { SetBit(RP04,RP07,RP02) }
  asm { SetBit(RP04,RP08,RP03) }
    if(tempBitField==-1)
      goto endSetForbidden9;
  asm { SetBit(RP03,RP06,RP00) }
  asm { SetBit(RP03,RP07,RP01) }
  asm { SetBit(RP03,RP08,RP02) }
  asm { SetBit(RP02,RP07,RP00) }
  asm { SetBit(RP02,RP08,RP01) }
  asm { SetBit(RP01,RP08,RP00) }

endSetForbidden9:
  if(tempBitField==-1)
    goto returnPoint8;
  bitField=tempBitField;
  bitFieldStore[9]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE9;

      if(n==10)       //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        count++;
        costasArrays[count*n]=(512>>RP00);
        costasArrays[count*n+1]=(512>>RP01);
        costasArrays[count*n+2]=(512>>RP02);
        costasArrays[count*n+3]=(512>>RP03);
        costasArrays[count*n+4]=(512>>RP04);
        costasArrays[count*n+5]=(512>>RP05);
        costasArrays[count*n+6]=(512>>RP06);
        costasArrays[count*n+7]=(512>>RP07);
        costasArrays[count*n+8]=(512>>RP08);
        costasArrays[count*n+9]=(512>>RP09);
        count++;
        goto doReturn9;
      }
      goto begin10;
    }
returnPoint9:
    bitNum++;
  }
doReturn9:
  // bitNum=RowPosStore[row];
  LOAD8;
  bitField=bitFieldStore[8];
  goto returnPoint8;

begin10:
  tempBitField=blankBitField;
setForbidden10:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm { HadBit(RP08)}
  asm {HadBit(RP09)}

  asm { SetBit(RP09,RP01,RP00) }
  asm { SetBit(RP09,RP02,RP01) }
  asm { SetBit(RP09,RP03,RP02) }
  asm { SetBit(RP09,RP04,RP03) }
  asm { SetBit(RP09,RP05,RP04) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP09,RP06,RP05) }
  asm { SetBit(RP09,RP07,RP06) }
  asm { SetBit(RP09,RP08,RP07) }
  asm { SetBit(RP09,RP09,RP08) }
  asm { SetBit(RP08,RP02,RP00) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP08,RP03,RP01) }
  asm { SetBit(RP08,RP04,RP02) }
  asm { SetBit(RP08,RP05,RP03) }
  asm { SetBit(RP08,RP06,RP04) }
  asm { SetBit(RP08,RP07,RP05) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP08,RP08,RP06) }
  asm { SetBit(RP08,RP09,RP07) }
  asm { SetBit(RP07,RP03,RP00) }
  asm { SetBit(RP07,RP04,RP01) }
  asm { SetBit(RP07,RP05,RP02) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP07,RP06,RP03) }
  asm { SetBit(RP07,RP07,RP04) }
  asm { SetBit(RP07,RP08,RP05) }
  asm { SetBit(RP07,RP09,RP06) }
  asm { SetBit(RP06,RP04,RP00) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP06,RP05,RP01) }
  asm { SetBit(RP06,RP06,RP02) }
  asm { SetBit(RP06,RP07,RP03) }
  asm { SetBit(RP06,RP08,RP04) }
  asm { SetBit(RP06,RP09,RP05) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP05,RP05,RP00) }
  asm { SetBit(RP05,RP06,RP01) }
  asm { SetBit(RP05,RP07,RP02) }
  asm { SetBit(RP05,RP08,RP03) }
  asm { SetBit(RP05,RP09,RP04) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP04,RP06,RP00) }
  asm { SetBit(RP04,RP07,RP01) }
  asm { SetBit(RP04,RP08,RP02) }
  asm { SetBit(RP04,RP09,RP03) }
  asm { SetBit(RP03,RP07,RP00) }
    if(tempBitField==-1)
      goto endSetForbidden10;
  asm { SetBit(RP03,RP08,RP01) }
  asm { SetBit(RP03,RP09,RP02) }
  asm { SetBit(RP02,RP08,RP00) }
  asm { SetBit(RP02,RP09,RP01) }
  asm { SetBit(RP01,RP09,RP00) }
```

```
endSetForbidden10:
  if(tempBitField==-1)
    goto returnPoint9;

  bitField=tempBitField;
  bitFieldStore[10]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE10;

      if(n==11)       //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        count++;
        costasArrays[count*n]=(1024>>RP00);
        costasArrays[count*n+1]=(1024>>RP01);
        costasArrays[count*n+2]=(1024>>RP02);
        costasArrays[count*n+3]=(1024>>RP03);
        costasArrays[count*n+4]=(1024>>RP04);
        costasArrays[count*n+5]=(1024>>RP05);
        costasArrays[count*n+6]=(1024>>RP06);
        costasArrays[count*n+7]=(1024>>RP07);
        costasArrays[count*n+8]=(1024>>RP08);
        costasArrays[count*n+9]=(1024>>RP09);
        costasArrays[count*n+10]=(1024>>RP10);
        count++;
        goto doReturn10;
      }
      goto begin11;
    }
returnPoint10:
    bitNum++;
  }
doReturn10:
  // bitNum=RowPosStore[row];
  LOAD9;
  bitField=bitFieldStore[9];
  goto returnPoint9;

begin11:
  tempBitField=blankBitField;
setForbidden11:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm { HadBit(RP08)}
  asm { HadBit(RP09)}
  asm {HadBit(RP10)}

  asm { SetBit(RP10,RP01,RP00) }
  asm { SetBit(RP10,RP02,RP01) }
  asm { SetBit(RP10,RP03,RP02) }
  asm { SetBit(RP10,RP04,RP03) }
  asm { SetBit(RP10,RP05,RP04) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP10,RP06,RP05) }
  asm { SetBit(RP10,RP07,RP06) }
  asm { SetBit(RP10,RP08,RP07) }
  asm { SetBit(RP10,RP09,RP08) }
  asm { SetBit(RP10,RP10,RP09) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP09,RP02,RP00) }
  asm { SetBit(RP09,RP03,RP01) }
  asm { SetBit(RP09,RP04,RP02) }
  asm { SetBit(RP09,RP05,RP03) }
  asm { SetBit(RP09,RP06,RP04) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP09,RP07,RP05) }
  asm { SetBit(RP09,RP08,RP06) }
  asm { SetBit(RP09,RP09,RP07) }
  asm { SetBit(RP09,RP10,RP08) }
  asm { SetBit(RP08,RP03,RP00) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP08,RP04,RP01) }
  asm { SetBit(RP08,RP05,RP02) }
  asm { SetBit(RP08,RP06,RP03) }
  asm { SetBit(RP08,RP07,RP04) }
  asm { SetBit(RP08,RP08,RP05) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP08,RP09,RP06) }
  asm { SetBit(RP08,RP10,RP07) }
  asm { SetBit(RP07,RP04,RP00) }
  asm { SetBit(RP07,RP05,RP01) }
  asm { SetBit(RP07,RP06,RP02) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP07,RP07,RP03) }
  asm { SetBit(RP07,RP08,RP04) }
  asm { SetBit(RP07,RP09,RP05) }
  asm { SetBit(RP07,RP10,RP06) }
  asm { SetBit(RP06,RP05,RP00) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP06,RP06,RP01) }
  asm { SetBit(RP06,RP07,RP02) }
  asm { SetBit(RP06,RP08,RP03) }
  asm { SetBit(RP06,RP09,RP04) }
  asm { SetBit(RP06,RP10,RP05) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP05,RP06,RP00) }
  asm { SetBit(RP05,RP07,RP01) }
  asm { SetBit(RP05,RP08,RP02) }
  asm { SetBit(RP05,RP09,RP03) }
  asm { SetBit(RP05,RP10,RP04) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP04,RP07,RP00) }
  asm { SetBit(RP04,RP08,RP01) }
  asm { SetBit(RP04,RP09,RP02) }
  asm { SetBit(RP04,RP10,RP03) }
  asm { SetBit(RP03,RP08,RP00) }
  if(tempBitField==-1) goto endSetForbidden11;
  asm { SetBit(RP03,RP09,RP01) }
  asm { SetBit(RP03,RP10,RP02) }
  asm { SetBit(RP02,RP09,RP00) }
  asm { SetBit(RP02,RP10,RP01) }
  asm { SetBit(RP01,RP10,RP00) }

endSetForbidden11:
  if(tempBitField==-1)
    goto returnPoint10;

  bitField=tempBitField;
  bitFieldStore[11]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE11;

      if(n==12)       //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        costasArrays[count*n+11]=(1<<RP11);
```

```
        count++;
        costasArrays[count*n]=(2048>>RP00);
        costasArrays[count*n+1]=(2048>>RP01);
        costasArrays[count*n+2]=(2048>>RP02);
        costasArrays[count*n+3]=(2048>>RP03);
        costasArrays[count*n+4]=(2048>>RP04);
        costasArrays[count*n+5]=(2048>>RP05);
        costasArrays[count*n+6]=(2048>>RP06);
        costasArrays[count*n+7]=(2048>>RP07);
        costasArrays[count*n+8]=(2048>>RP08);
        costasArrays[count*n+9]=(2048>>RP09);
        costasArrays[count*n+10]=(2048>>RP10);
        costasArrays[count*n+11]=(2048>>RP11);
        count++;
        goto doReturn11;
      }
      goto begin12;
    }
returnPoint11:
    bitNum++;
  }
doReturn11:
  // bitNum=RowPosStore[row];
  LOAD10;
  bitField=bitFieldStore[10];
  goto returnPoint10;

begin12:
  tempBitField=blankBitField;
setForbidden12:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm { HadBit(RP08)}
  asm { HadBit(RP09)}
  asm {HadBit(RP10)}
  asm {HadBit(RP11)}

  asm { SetBit(RP11,RP01,RP00) }
  asm { SetBit(RP11,RP02,RP01) }
  asm { SetBit(RP11,RP03,RP02) }
  asm { SetBit(RP11,RP04,RP03) }
  asm { SetBit(RP11,RP05,RP04) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP11,RP06,RP05) }
  asm { SetBit(RP11,RP07,RP06) }
  asm { SetBit(RP11,RP08,RP07) }
  asm { SetBit(RP11,RP09,RP08) }
  asm { SetBit(RP11,RP10,RP09) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP11,RP11,RP10) }
  asm { SetBit(RP10,RP02,RP00) }
  asm { SetBit(RP10,RP03,RP01) }
  asm { SetBit(RP10,RP04,RP02) }
  asm { SetBit(RP10,RP05,RP03) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP10,RP06,RP04) }
  asm { SetBit(RP10,RP07,RP05) }
  asm { SetBit(RP10,RP08,RP06) }
  asm { SetBit(RP10,RP09,RP07) }
  asm { SetBit(RP10,RP10,RP08) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP10,RP11,RP09) }
  asm { SetBit(RP09,RP03,RP00) }
  asm { SetBit(RP09,RP04,RP01) }
  asm { SetBit(RP09,RP05,RP02) }
  asm { SetBit(RP09,RP06,RP03) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP09,RP07,RP04) }
  asm { SetBit(RP09,RP08,RP05) }
  asm { SetBit(RP09,RP09,RP06) }
  asm { SetBit(RP09,RP10,RP07) }
  asm { SetBit(RP09,RP11,RP08) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP08,RP04,RP00) }
  asm { SetBit(RP08,RP05,RP01) }
  asm { SetBit(RP08,RP06,RP02) }
  asm { SetBit(RP08,RP07,RP03) }
  asm { SetBit(RP08,RP08,RP04) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP08,RP09,RP05) }
  asm { SetBit(RP08,RP10,RP06) }
  asm { SetBit(RP08,RP11,RP07) }
  asm { SetBit(RP07,RP05,RP00) }
  asm { SetBit(RP07,RP06,RP01) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP07,RP07,RP02) }
  asm { SetBit(RP07,RP08,RP03) }
  asm { SetBit(RP07,RP09,RP04) }
  asm { SetBit(RP07,RP10,RP05) }
  asm { SetBit(RP07,RP11,RP06) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP06,RP06,RP00) }
  asm { SetBit(RP06,RP07,RP01) }
  asm { SetBit(RP06,RP08,RP02) }
  asm { SetBit(RP06,RP09,RP03) }
  asm { SetBit(RP06,RP10,RP04) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP06,RP11,RP05) }
  asm { SetBit(RP05,RP07,RP00) }
  asm { SetBit(RP05,RP08,RP01) }
  asm { SetBit(RP05,RP09,RP02) }
  asm { SetBit(RP05,RP10,RP03) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP05,RP11,RP04) }
  asm { SetBit(RP04,RP08,RP00) }
  asm { SetBit(RP04,RP09,RP01) }
  asm { SetBit(RP04,RP10,RP02) }
  asm { SetBit(RP04,RP11,RP03) }
  if(tempBitField==-1) goto endSetForbidden12;
  asm { SetBit(RP03,RP09,RP00) }
  asm { SetBit(RP03,RP10,RP01) }
  asm { SetBit(RP03,RP11,RP02) }
  asm { SetBit(RP02,RP10,RP00) }
  asm { SetBit(RP02,RP11,RP01) }
  asm { SetBit(RP01,RP11,RP00) }

endSetForbidden12:
  if(tempBitField==-1)
    goto returnPoint11;

  bitField=tempBitField;
  bitFieldStore[12]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE12;

      if(n==13)       //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        costasArrays[count*n+11]=(1<<RP11);
        costasArrays[count*n+12]=(1<<RP12);
        count++;
        costasArrays[count*n]=(4096>>RP00);
        costasArrays[count*n+1]=(4096>>RP01);
        costasArrays[count*n+2]=(4096>>RP02);
        costasArrays[count*n+3]=(4096>>RP03);
        costasArrays[count*n+4]=(4096>>RP04);
        costasArrays[count*n+5]=(4096>>RP05);
        costasArrays[count*n+6]=(4096>>RP06);
        costasArrays[count*n+7]=(4096>>RP07);
        costasArrays[count*n+8]=(4096>>RP08);
        costasArrays[count*n+9]=(4096>>RP09);
        costasArrays[count*n+10]=(4096>>RP10);
        costasArrays[count*n+11]=(4096>>RP11);
        costasArrays[count*n+12]=(4096>>RP12);
```

```
        count++;
        goto doReturn12;
      }
      goto begin13;
    }
returnPoint12:
    bitNum++;
  }
doReturn12:
  // bitNum=RowPosStore[row];
  LOAD11:
  bitField=bitFieldStore[11];
  goto returnPoint11;

begin13:
  tempBitField=blankBitField;
setForbidden13:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm { HadBit(RP08)}
  asm { HadBit(RP09)}
  asm { HadBit(RP10)}
  asm { HadBit(RP11)}
  asm { HadBit(RP12)}
  asm { SetBit(RP12,RP01,RP00) }
  asm { SetBit(RP12,RP02,RP01) }
  asm { SetBit(RP12,RP03,RP02) }
  asm { SetBit(RP12,RP04,RP03) }
  asm { SetBit(RP12,RP05,RP04) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP12,RP06,RP05) }
  asm { SetBit(RP12,RP07,RP06) }
  asm { SetBit(RP12,RP08,RP07) }
  asm { SetBit(RP12,RP09,RP08) }
  asm { SetBit(RP12,RP10,RP09) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP12,RP11,RP10) }
  asm { SetBit(RP12,RP12,RP11) }
  asm { SetBit(RP11,RP02,RP00) }
  asm { SetBit(RP11,RP03,RP01) }
  asm { SetBit(RP11,RP04,RP02) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP11,RP05,RP03) }
  asm { SetBit(RP11,RP06,RP04) }
  asm { SetBit(RP11,RP07,RP05) }
  asm { SetBit(RP11,RP08,RP06) }
  asm { SetBit(RP11,RP09,RP07) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP11,RP10,RP08) }
  asm { SetBit(RP11,RP11,RP09) }
  asm { SetBit(RP11,RP12,RP10) }
  asm { SetBit(RP10,RP03,RP00) }
  asm { SetBit(RP10,RP04,RP01) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP10,RP05,RP02) }
  asm { SetBit(RP10,RP06,RP03) }
  asm { SetBit(RP10,RP07,RP04) }
  asm { SetBit(RP10,RP08,RP05) }
  asm { SetBit(RP10,RP09,RP06) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP10,RP10,RP07) }
  asm { SetBit(RP10,RP11,RP08) }
  asm { SetBit(RP10,RP12,RP09) }
  asm { SetBit(RP09,RP04,RP00) }
  asm { SetBit(RP09,RP05,RP01) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP09,RP06,RP02) }
  asm { SetBit(RP09,RP07,RP03) }
  asm { SetBit(RP09,RP08,RP04) }
  asm { SetBit(RP09,RP09,RP05) }
  asm { SetBit(RP09,RP10,RP06) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP09,RP11,RP07) }
  asm { SetBit(RP09,RP12,RP08) }
  asm { SetBit(RP08,RP05,RP00) }
  asm { SetBit(RP08,RP06,RP01) }
  asm { SetBit(RP08,RP07,RP02) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP08,RP08,RP03) }
  asm { SetBit(RP08,RP09,RP04) }
  asm { SetBit(RP08,RP10,RP05) }
  asm { SetBit(RP08,RP11,RP06) }
  asm { SetBit(RP08,RP12,RP07) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP07,RP06,RP00) }
  asm { SetBit(RP07,RP07,RP01) }
  asm { SetBit(RP07,RP08,RP02) }
  asm { SetBit(RP07,RP09,RP03) }
  asm { SetBit(RP07,RP10,RP04) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP07,RP11,RP05) }
  asm { SetBit(RP07,RP12,RP06) }
  asm { SetBit(RP06,RP07,RP00) }
  asm { SetBit(RP06,RP08,RP01) }
  asm { SetBit(RP06,RP09,RP02) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP06,RP10,RP03) }
  asm { SetBit(RP06,RP11,RP04) }
  asm { SetBit(RP06,RP12,RP05) }
  asm { SetBit(RP05,RP08,RP00) }
  asm { SetBit(RP05,RP09,RP01) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP05,RP10,RP02) }
  asm { SetBit(RP05,RP11,RP03) }
  asm { SetBit(RP05,RP12,RP04) }
  asm { SetBit(RP04,RP09,RP00) }
  asm { SetBit(RP04,RP10,RP01) }
  asm { SetBit(RP04,RP11,RP02) }
  if(tempBitField==-1) goto endSetForbidden13;
  asm { SetBit(RP04,RP12,RP03) }
  asm { SetBit(RP03,RP10,RP00) }
  asm { SetBit(RP03,RP11,RP01) }
  asm { SetBit(RP03,RP12,RP02) }
  asm { SetBit(RP02,RP11,RP00) }
  asm { SetBit(RP02,RP12,RP01) }
  asm { SetBit(RP01,RP12,RP00) }

endSetForbidden13:

  if(tempBitField==-1)
    goto returnPoint12;

  bitField=tempBitField;
  bitFieldStore[13]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE13:

      if(n==14)      //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        costasArrays[count*n+11]=(1<<RP11);
        costasArrays[count*n+12]=(1<<RP12);
        costasArrays[count*n+13]=(1<<RP13);
        count++;
        costasArrays[count*n]=(8192>>RP00);
        costasArrays[count*n+1]=(8192>>RP01);
        costasArrays[count*n+2]=(8192>>RP02);
        costasArrays[count*n+3]=(8192>>RP03);
        costasArrays[count*n+4]=(8192>>RP04);
        costasArrays[count*n+5]=(8192>>RP05);
        costasArrays[count*n+6]=(8192>>RP06);
        costasArrays[count*n+7]=(8192>>RP07);
        costasArrays[count*n+8]=(8192>>RP08);
```

```
        costasArrays[count*n+9]=(8192>>RP09);
        costasArrays[count*n+10]=(8192>>RP10);
        costasArrays[count*n+11]=(8192>>RP11);
        costasArrays[count*n+12]=(8192>>RP12);
        costasArrays[count*n+13]=(8192>>RP13);
        count++;
        goto doReturn13;
      }
      goto begin14;
    }
returnPoint13:
    bitNum++;
  }
doReturn13:
  // bitNum=RowPosStore[row];
  LOAD12:
  bitField=bitFieldStore[12];
  goto returnPoint12;

begin14:
  tempBitField=blankBitField;
setForbidden14:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm { HadBit(RP08)}
  asm { HadBit(RP09)}
  asm { HadBit(RP10)}
  asm { HadBit(RP11)}
  asm { HadBit(RP12)}
  asm { HadBit(RP13)}
  asm { SetBit(RP13,RP01,RP00) }
  asm { SetBit(RP13,RP02,RP01) }
  asm { SetBit(RP13,RP03,RP02) }
  asm { SetBit(RP13,RP04,RP03) }
  asm { SetBit(RP13,RP05,RP04) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP13,RP06,RP05) }
  asm { SetBit(RP13,RP07,RP06) }
  asm { SetBit(RP13,RP08,RP07) }
  asm { SetBit(RP13,RP09,RP08) }
  asm { SetBit(RP13,RP10,RP09) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP13,RP11,RP10) }
  asm { SetBit(RP13,RP12,RP11) }
  asm { SetBit(RP13,RP13,RP12) }
  asm { SetBit(RP12,RP02,RP00) }
  asm { SetBit(RP12,RP03,RP01) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP12,RP04,RP02) }
  asm { SetBit(RP12,RP05,RP03) }
  asm { SetBit(RP12,RP06,RP04) }
  asm { SetBit(RP12,RP07,RP05) }
  asm { SetBit(RP12,RP08,RP06) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP12,RP09,RP07) }
  asm { SetBit(RP12,RP10,RP08) }
  asm { SetBit(RP12,RP11,RP09) }
  asm { SetBit(RP12,RP12,RP10) }
  asm { SetBit(RP12,RP13,RP11) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP11,RP03,RP00) }
  asm { SetBit(RP11,RP04,RP01) }
  asm { SetBit(RP11,RP05,RP02) }
  asm { SetBit(RP11,RP06,RP03) }
  asm { SetBit(RP11,RP07,RP04) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP11,RP08,RP05) }
  asm { SetBit(RP11,RP09,RP06) }
  asm { SetBit(RP11,RP10,RP07) }
  asm { SetBit(RP11,RP11,RP08) }
  asm { SetBit(RP11,RP12,RP09) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP11,RP13,RP10) }
  asm { SetBit(RP10,RP04,RP00) }
  asm { SetBit(RP10,RP05,RP01) }
  asm { SetBit(RP10,RP06,RP02) }
  asm { SetBit(RP10,RP07,RP03) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP10,RP08,RP04) }
  asm { SetBit(RP10,RP09,RP05) }
  asm { SetBit(RP10,RP10,RP06) }
  asm { SetBit(RP10,RP11,RP07) }
  asm { SetBit(RP10,RP12,RP08) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP10,RP13,RP09) }
  asm { SetBit(RP09,RP05,RP00) }
  asm { SetBit(RP09,RP06,RP01) }
  asm { SetBit(RP09,RP07,RP02) }
  asm { SetBit(RP09,RP08,RP03) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP09,RP09,RP04) }
  asm { SetBit(RP09,RP10,RP05) }
  asm { SetBit(RP09,RP11,RP06) }
  asm { SetBit(RP09,RP12,RP07) }
  asm { SetBit(RP09,RP13,RP08) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP08,RP06,RP00) }
  asm { SetBit(RP08,RP07,RP01) }
  asm { SetBit(RP08,RP08,RP02) }
  asm { SetBit(RP08,RP09,RP03) }
  asm { SetBit(RP08,RP10,RP04) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP08,RP11,RP05) }
  asm { SetBit(RP08,RP12,RP06) }
  asm { SetBit(RP08,RP13,RP07) }
  asm { SetBit(RP07,RP07,RP00) }
  asm { SetBit(RP07,RP08,RP01) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP07,RP09,RP02) }
  asm { SetBit(RP07,RP10,RP03) }
  asm { SetBit(RP07,RP11,RP04) }
  asm { SetBit(RP07,RP12,RP05) }
  asm { SetBit(RP07,RP13,RP06) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP06,RP08,RP00) }
  asm { SetBit(RP06,RP09,RP01) }
  asm { SetBit(RP06,RP10,RP02) }
  asm { SetBit(RP06,RP11,RP03) }
  asm { SetBit(RP06,RP12,RP04) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP06,RP13,RP05) }
  asm { SetBit(RP05,RP09,RP00) }
  asm { SetBit(RP05,RP10,RP01) }
  asm { SetBit(RP05,RP11,RP02) }
  asm { SetBit(RP05,RP12,RP03) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP05,RP13,RP04) }
  asm { SetBit(RP04,RP10,RP00) }
  asm { SetBit(RP04,RP11,RP01) }
  asm { SetBit(RP04,RP12,RP02) }
  asm { SetBit(RP04,RP13,RP03) }
  if(tempBitField==-1) goto endSetForbidden14;
  asm { SetBit(RP03,RP11,RP00) }
  asm { SetBit(RP03,RP12,RP01) }
  asm { SetBit(RP03,RP13,RP02) }
  asm { SetBit(RP02,RP12,RP00) }
  asm { SetBit(RP02,RP13,RP01) }
  asm { SetBit(RP01,RP13,RP00) }

endSetForbidden14:

  if(tempBitField==-1)
    goto returnPoint13;

  bitField=tempBitField;
  bitFieldStore[14]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE14:

      if(n==15)       //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
```

```
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        costasArrays[count*n+11]=(1<<RP11);
        costasArrays[count*n+12]=(1<<RP12);
        costasArrays[count*n+13]=(1<<RP13);
        costasArrays[count*n+14]=(1<<RP14);
        count++;
        costasArrays[count*n]=(16384>>RP00);
        costasArrays[count*n+1]=(16384>>RP01);
        costasArrays[count*n+2]=(16384>>RP02);
        costasArrays[count*n+3]=(16384>>RP03);
        costasArrays[count*n+4]=(16384>>RP04);
        costasArrays[count*n+5]=(16384>>RP05);
        costasArrays[count*n+6]=(16384>>RP06);
        costasArrays[count*n+7]=(16384>>RP07);
        costasArrays[count*n+8]=(16384>>RP08);
        costasArrays[count*n+9]=(16384>>RP09);
        costasArrays[count*n+10]=(16384>>RP10);
        costasArrays[count*n+11]=(16384>>RP11);
        costasArrays[count*n+12]=(16384>>RP12);
        costasArrays[count*n+13]=(16384>>RP13);
        costasArrays[count*n+14]=(16384>>RP14);
        count++;
        goto doReturn14;
      }
      goto begin15;
    }
returnPoint14:
    bitNum++;
  }
doReturn14:
  // bitNum=RowPosStore[row];
  LOAD13:
  bitField=bitFieldStore[13];
  goto returnPoint13;

begin15:
  tempBitField=blankBitField;
setForbidden15:
  asm { HadBit(RP00)}
  asm { HadBit(RP01)}
  asm { HadBit(RP02)}
  asm { HadBit(RP03)}
  asm { HadBit(RP04)}
  asm { HadBit(RP05)}
  asm { HadBit(RP06)}
  asm { HadBit(RP07)}
  asm { HadBit(RP08)}
  asm { HadBit(RP09)}
  asm { HadBit(RP10)}
  asm { HadBit(RP11)}
  asm { HadBit(RP12)}
  asm { HadBit(RP13)}
  asm { HadBit(RP14)}
  asm { SetBit(RP14,RP01,RP00) }
  asm { SetBit(RP14,RP02,RP01) }
  asm { SetBit(RP14,RP03,RP02) }
  asm { SetBit(RP14,RP04,RP03) }
  asm { SetBit(RP14,RP05,RP04) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP14,RP06,RP05) }
  asm { SetBit(RP14,RP07,RP06) }
  asm { SetBit(RP14,RP08,RP07) }
  asm { SetBit(RP14,RP09,RP08) }
  asm { SetBit(RP14,RP10,RP09) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP14,RP11,RP10) }
  asm { SetBit(RP14,RP12,RP11) }
  asm { SetBit(RP14,RP13,RP12) }
  asm { SetBit(RP14,RP14,RP13) }
  asm { SetBit(RP13,RP02,RP00) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP13,RP03,RP01) }
  asm { SetBit(RP13,RP04,RP02) }
  asm { SetBit(RP13,RP05,RP03) }
  asm { SetBit(RP13,RP06,RP04) }
  asm { SetBit(RP13,RP07,RP05) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP13,RP08,RP06) }
  asm { SetBit(RP13,RP09,RP07) }
  asm { SetBit(RP13,RP10,RP08) }
  asm { SetBit(RP13,RP11,RP09) }
  asm { SetBit(RP13,RP12,RP10) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP13,RP13,RP11) }
  asm { SetBit(RP13,RP14,RP12) }
  asm { SetBit(RP12,RP03,RP00) }
  asm { SetBit(RP12,RP04,RP01) }
  asm { SetBit(RP12,RP05,RP02) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP12,RP06,RP03) }
  asm { SetBit(RP12,RP07,RP04) }
  asm { SetBit(RP12,RP08,RP05) }
  asm { SetBit(RP12,RP09,RP06) }
  asm { SetBit(RP12,RP10,RP07) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP12,RP11,RP08) }
  asm { SetBit(RP12,RP12,RP09) }
  asm { SetBit(RP12,RP13,RP10) }
  asm { SetBit(RP12,RP14,RP11) }
  asm { SetBit(RP11,RP04,RP00) }
  if(tempBitField== 1) goto endSetForbidden15;
  asm { SetBit(RP11,RP05,RP01) }
  asm { SetBit(RP11,RP06,RP02) }
  asm { SetBit(RP11,RP07,RP03) }
  asm { SetBit(RP11,RP08,RP04) }
  asm { SetBit(RP11,RP09,RP05) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP11,RP10,RP06) }
  asm { SetBit(RP11,RP11,RP07) }
  asm { SetBit(RP11,RP12,RP08) }
  asm { SetBit(RP11,RP13,RP09) }
  asm { SetBit(RP11,RP14,RP10) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP10,RP05,RP00) }
  asm { SetBit(RP10,RP06,RP01) }
  asm { SetBit(RP10,RP07,RP02) }
  asm { SetBit(RP10,RP08,RP03) }
  asm { SetBit(RP10,RP09,RP04) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP10,RP10,RP05) }
  asm { SetBit(RP10,RP11,RP06) }
  asm { SetBit(RP10,RP12,RP07) }
  asm { SetBit(RP10,RP13,RP08) }
  asm { SetBit(RP10,RP14,RP09) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP09,RP06,RP00) }
  asm { SetBit(RP09,RP07,RP01) }
  asm { SetBit(RP09,RP08,RP02) }
  asm { SetBit(RP09,RP09,RP03) }
  asm { SetBit(RP09,RP10,RP04) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP09,RP11,RP05) }
  asm { SetBit(RP09,RP12,RP06) }
  asm { SetBit(RP09,RP13,RP07) }
  asm { SetBit(RP09,RP14,RP08) }
  asm { SetBit(RP08,RP07,RP00) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP08,RP08,RP01) }
  asm { SetBit(RP08,RP09,RP02) }
  asm { SetBit(RP08,RP10,RP03) }
  asm { SetBit(RP08,RP11,RP04) }
  asm { SetBit(RP08,RP12,RP05) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP08,RP13,RP06) }
  asm { SetBit(RP08,RP14,RP07) }
  asm { SetBit(RP07,RP08,RP00) }
  asm { SetBit(RP07,RP09,RP01) }
  asm { SetBit(RP07,RP10,RP02) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP07,RP11,RP03) }
  asm { SetBit(RP07,RP12,RP04) }
  asm { SetBit(RP07,RP13,RP05) }
  asm { SetBit(RP07,RP14,RP06) }
  asm { SetBit(RP06,RP09,RP00) }
```

```
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP06,RP10,RP01) }
  asm { SetBit(RP06,RP11,RP02) }
  asm { SetBit(RP06,RP12,RP03) }
  asm { SetBit(RP06,RP13,RP04) }
  asm { SetBit(RP06,RP14,RP05) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP05,RP10,RP00) }
  asm { SetBit(RP05,RP11,RP01) }
  asm { SetBit(RP05,RP12,RP02) }
  asm { SetBit(RP05,RP13,RP03) }
  asm { SetBit(RP05,RP14,RP04) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP04,RP11,RP00) }
  asm { SetBit(RP04,RP12,RP01) }
  asm { SetBit(RP04,RP13,RP02) }
  asm { SetBit(RP04,RP14,RP03) }
  asm { SetBit(RP03,RP12,RP00) }
  if(tempBitField==-1) goto endSetForbidden15;
  asm { SetBit(RP03,RP13,RP01) }
  asm { SetBit(RP03,RP14,RP02) }
  asm { SetBit(RP02,RP13,RP00) }
  asm { SetBit(RP02,RP14,RP01) }
  asm { SetBit(RP01,RP14,RP00) }

endSetForbidden15:

  if(tempBitField==-1)
    goto returnPoint14;

  bitField=tempBitField;
  bitFieldStore[15]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;
      STORE15:

      if(n==16)       //record layout!
      {
        if(RP00>=n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        costasArrays[count*n+11]=(1<<RP11);
        costasArrays[count*n+12]=(1<<RP12);
        costasArrays[count*n+13]=(1<<RP13);
        costasArrays[count*n+14]=(1<<RP14);
        costasArrays[count*n+15]=(1<<RP15);
        count++;
        costasArrays[count*n]=(32768>>RP00);
        costasArrays[count*n+1]=(32768>>RP01);
        costasArrays[count*n+2]=(32768>>RP02);
        costasArrays[count*n+3]=(32768>>RP03);
        costasArrays[count*n+4]=(32768>>RP04);
        costasArrays[count*n+5]=(32768>>RP05);
        costasArrays[count*n+6]=(32768>>RP06);
        costasArrays[count*n+7]=(32768>>RP07);
        costasArrays[count*n+8]=(32768>>RP08);
        costasArrays[count*n+9]=(32768>>RP09);
        costasArrays[count*n+10]=(32768>>RP10);
        costasArrays[count*n+11]=(32768>>RP11);
        costasArrays[count*n+12]=(32768>>RP12);
        costasArrays[count*n+13]=(32768>>RP13);
        costasArrays[count*n+14]=(32768>>RP14);
        costasArrays[count*n+15]=(32768>>RP15);
        count++;
        goto doReturn15;
      }
      goto begin16;
    }

returnPoint15:
      bitNum++;
    }
doReturn15:
    // bitNum=RowPosStore[row];
    LOAD14:
    bitField=bitFieldStore[14];
    goto returnPoint14;

begin16:
    tempBitField=blankBitField;
setForbidden16:
    asm { HadBit(RP00)}
    asm { HadBit(RP01)}
    asm { HadBit(RP02)}
    asm { HadBit(RP03)}
    asm { HadBit(RP04)}
    asm { HadBit(RP05)}
    asm { HadBit(RP06)}
    asm { HadBit(RP07)}
    asm { HadBit(RP08)}
    asm { HadBit(RP09)}
    asm { HadBit(RP10)}
    asm { HadBit(RP11)}
    asm { HadBit(RP12)}
    asm { HadBit(RP13)}
    asm { HadBit(RP14)}
    asm { HadBit(RP15)}
    asm { SetBit(RP15,RP01,RP00) }
    asm { SetBit(RP15,RP02,RP01) }
    asm { SetBit(RP15,RP03,RP02) }
    asm { SetBit(RP15,RP04,RP03) }
    asm { SetBit(RP15,RP05,RP04) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP15,RP06,RP05) }
    asm { SetBit(RP15,RP07,RP06) }
    asm { SetBit(RP15,RP08,RP07) }
    asm { SetBit(RP15,RP09,RP08) }
    asm { SetBit(RP15,RP10,RP09) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP15,RP11,RP10) }
    asm { SetBit(RP15,RP12,RP11) }
    asm { SetBit(RP15,RP13,RP12) }
    asm { SetBit(RP15,RP14,RP13) }
    asm { SetBit(RP15,RP15,RP14) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP14,RP02,RP00) }
    asm { SetBit(RP14,RP03,RP01) }
    asm { SetBit(RP14,RP04,RP02) }
    asm { SetBit(RP14,RP05,RP03) }
    asm { SetBit(RP14,RP06,RP04) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP14,RP07,RP05) }
    asm { SetBit(RP14,RP08,RP06) }
    asm { SetBit(RP14,RP09,RP07) }
    asm { SetBit(RP14,RP10,RP08) }
    asm { SetBit(RP14,RP11,RP09) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP14,RP12,RP10) }
    asm { SetBit(RP14,RP13,RP11) }
    asm { SetBit(RP14,RP14,RP12) }
    asm { SetBit(RP14,RP15,RP13) }
    asm { SetBit(RP13,RP03,RP00) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP13,RP04,RP01) }
    asm { SetBit(RP13,RP05,RP02) }
    asm { SetBit(RP13,RP06,RP03) }
    asm { SetBit(RP13,RP07,RP04) }
    asm { SetBit(RP13,RP08,RP05) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP13,RP09,RP06) }
    asm { SetBit(RP13,RP10,RP07) }
    asm { SetBit(RP13,RP11,RP08) }
    asm { SetBit(RP13,RP12,RP09) }
    asm { SetBit(RP13,RP13,RP10) }
    if(tempBitField==-1) goto endSetForbidden16;
    asm { SetBit(RP13,RP14,RP11) }
    asm { SetBit(RP13,RP15,RP12) }
    asm { SetBit(RP12,RP04,RP00) }
    asm { SetBit(RP12,RP05,RP01) }
    asm { SetBit(RP12,RP06,RP02) }
    if(tempBitField==-1) goto endSetForbidden16;
```

```
asm { SetBit(RP12,RP07,RP03) }
asm { SetBit(RP12,RP08,RP04) }
asm { SetBit(RP12,RP09,RP05) }
asm { SetBit(RP12,RP10,RP06) }
asm { SetBit(RP12,RP11,RP07) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP12,RP12,RP08) }
asm { SetBit(RP12,RP13,RP09) }
asm { SetBit(RP12,RP14,RP10) }
asm { SetBit(RP12,RP15,RP11) }
asm { SetBit(RP11,RP05,RP00) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP11,RP06,RP01) }
asm { SetBit(RP11,RP07,RP02) }
asm { SetBit(RP11,RP08,RP03) }
asm { SetBit(RP11,RP09,RP04) }
asm { SetBit(RP11,RP10,RP05) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP11,RP11,RP06) }
asm { SetBit(RP11,RP12,RP07) }
asm { SetBit(RP11,RP13,RP08) }
asm { SetBit(RP11,RP14,RP09) }
asm { SetBit(RP11,RP15,RP10) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP10,RP06,RP00) }
asm { SetBit(RP10,RP07,RP01) }
asm { SetBit(RP10,RP08,RP02) }
asm { SetBit(RP10,RP09,RP03) }
asm { SetBit(RP10,RP10,RP04) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP10,RP11,RP05) }
asm { SetBit(RP10,RP12,RP06) }
asm { SetBit(RP10,RP13,RP07) }
asm { SetBit(RP10,RP14,RP08) }
asm { SetBit(RP10,RP15,RP09) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP09,RP07,RP00) }
asm { SetBit(RP09,RP08,RP01) }
asm { SetBit(RP09,RP09,RP02) }
asm { SetBit(RP09,RP10,RP03) }
asm { SetBit(RP09,RP11,RP04) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP09,RP12,RP05) }
asm { SetBit(RP09,RP13,RP06) }
asm { SetBit(RP09,RP14,RP07) }
asm { SetBit(RP09,RP15,RP08) }
asm { SetBit(RP08,RP08,RP00) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP08,RP09,RP01) }
asm { SetBit(RP08,RP10,RP02) }
asm { SetBit(RP08,RP11,RP03) }
asm { SetBit(RP08,RP12,RP04) }
asm { SetBit(RP08,RP13,RP05) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP08,RP14,RP06) }
asm { SetBit(RP08,RP15,RP07) }
asm { SetBit(RP07,RP09,RP00) }
asm { SetBit(RP07,RP10,RP01) }
asm { SetBit(RP07,RP11,RP02) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP07,RP12,RP03) }
asm { SetBit(RP07,RP13,RP04) }
asm { SetBit(RP07,RP14,RP05) }
asm { SetBit(RP07,RP15,RP06) }
asm { SetBit(RP06,RP10,RP00) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP06,RP11,RP01) }
asm { SetBit(RP06,RP12,RP02) }
asm { SetBit(RP06,RP13,RP03) }
asm { SetBit(RP06,RP14,RP04) }
asm { SetBit(RP06,RP15,RP05) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP05,RP11,RP00) }
asm { SetBit(RP05,RP12,RP01) }
asm { SetBit(RP05,RP13,RP02) }
asm { SetBit(RP05,RP14,RP03) }
asm { SetBit(RP05,RP15,RP04) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP04,RP12,RP00) }
asm { SetBit(RP04,RP13,RP01) }
asm { SetBit(RP04,RP14,RP02) }
asm { SetBit(RP04,RP15,RP03) }
asm { SetBit(RP03,RP13,RP00) }
if(tempBitField==-1) goto endSetForbidden16;
asm { SetBit(RP03,RP14,RP01) }
asm { SetBit(RP03,RP15,RP02) }
asm { SetBit(RP02,RP14,RP00) }
asm { SetBit(RP02,RP15,RP01) }
asm { SetBit(RP01,RP15,RP00) }

endSetForbidden16:

  if(tempBitField== 1)
    goto returnPoint15;

  bitField=tempBitField;
  bitFieldStore[16]=bitField;

  for(bitNum=0;bitNum<n;)
  {
    if((bitField&(kOne<<bitNum))==0)
    {
      //RowPosStore[row]=bitNum;

      if(n==17)       //record layout!
      {
        if(RP01>n/2 && RP00==n/2)
          return count;
        costasArrays[count*n]=(1<<RP00);
        costasArrays[count*n+1]=(1<<RP01);
        costasArrays[count*n+2]=(1<<RP02);
        costasArrays[count*n+3]=(1<<RP03);
        costasArrays[count*n+4]=(1<<RP04);
        costasArrays[count*n+5]=(1<<RP05);
        costasArrays[count*n+6]=(1<<RP06);
        costasArrays[count*n+7]=(1<<RP07);
        costasArrays[count*n+8]=(1<<RP08);
        costasArrays[count*n+9]=(1<<RP09);
        costasArrays[count*n+10]=(1<<RP10);
        costasArrays[count*n+11]=(1<<RP11);
        costasArrays[count*n+12]=(1<<RP12);
        costasArrays[count*n+13]=(1<<RP13);
        costasArrays[count*n+14]=(1<<RP14);
        costasArrays[count*n+15]=(1<<RP15);
        costasArrays[count*n+16]=(1<<bitNum);
        count++;
        costasArrays[count*n]=(65536>>RP00);
        costasArrays[count*n+1]=(65536>>RP01);
        costasArrays[count*n+2]=(65536>>RP02);
        costasArrays[count*n+3]=(65536>>RP03);
        costasArrays[count*n+4]=(65536>>RP04);
        costasArrays[count*n+5]=(65536>>RP05);
        costasArrays[count*n+6]=(65536>>RP06);
        costasArrays[count*n+7]=(65536>>RP07);
        costasArrays[count*n+8]=(65536>>RP08);
        costasArrays[count*n+9]=(65536>>RP09);
        costasArrays[count*n+10]=(65536>>RP10);
        costasArrays[count*n+11]=(65536>>RP11);
        costasArrays[count*n+12]=(65536>>RP12);
        costasArrays[count*n+13]=(65536>>RP13);
        costasArrays[count*n+14]=(65536>>RP14);
        costasArrays[count*n+15]=(65536>>RP15);
        costasArrays[count*n+16]=(65536>>bitNum);
        count++;
        goto doReturn16;
      }
      return count;
    }
  returnPoint16:
    bitNum++;
  }
  doReturn16:
    // bitNum=RowPosStore[row];
    LOAD15:
    bitField=bitFieldStore[15];
    goto returnPoint15;
  }
```

By MacTech's Production Staff

Large Format Printing

The HP 2500CP Series Printer

For years, large format (e.g., poster size) printers have been available. Recently, there have been some new developments in these types of printers that warrant a look if you have any large format needs. The question that we asked ourselves were "are these the types of printers that we'd want to have in a typical network?" To answer this, we looked at two sets of solutions — low end (starting at around $1700) and high end. Here's what we found.

Choices

There are a wide range of choices that you have when selecting these printers. First, you need to determine your needs. For example, what types of media (paper) do you need? What about ink? (Some inks will fade quickly with any light exposure). What resolution? What are the types of color support that you need?

It's easy to get buried in these details — but you will need to determine them in order to select the right printer.

HP DesignJet 2500CP

This photo-quality color printer has changed our opinion of large format printing. We took the HP DesignJet 2500CP out of the box, plugged it into a Mac G3, and it was ready to go. The DesignJet 2500CP and 3500CP series both come with Adobe PostScript drivers, built-in RIP, and a 4.3 GB hard disk ready to connect to Ethernet.

As far as color matching goes, HP's integrated system of printer, ink, papers and films allows the HP DesignJet 2500CP and 3500CP series printers to produce high quality images with millions of colors. Automatic color calibration guarantees high-quality output every time, and is PANTONE compatible.

We did a test print to check out the saturation, and was extremely pleased at the results. Some designs can be heavy in black saturation, resulting in banding on some printers, but not the DesignJet 2500CP. The completely black and shadowed areas of the print were solid and clear, and the midtones and highlights were equally as clear, with beautiful color saturation on the glossy paper.

This system is very easy to run from your desktop PC. Selecting drivers, changing printing settings, calibration, and everything else involved in getting your design from your monitor to larger-than-life, photo-quality print size is as easy as working your desktop ink jet printer. All at a cost of about 40 cents a square foot on coated paper. I would definitely recommend the HP DesignJet (2500CP?) for anyone serious about printing quality large format prints from their desktop PC.

Choice of Ink and Media/Cost Per Copy

With their large-capacity ink systems, the DesignJet 2500CP printer offers very easy switching between two ink sets — dye-based for maximum image quality, or pigment-based for enhanced lightfastness. A choice of specially-developed HP and 3M DesignJet media contributes further to the output quality.

The HP DesignJet 2500CP printer has a low cost-per-copy, and is an excellent addition to anyone looking for an affordable alternative to large format printing.

Features

- True 600-dpi black and color print resolution
- Photo-quality output in over 16 million colors
- A1/D-size color images in around 8 minutes
- Large-capacity ink system — up to 2000 sq. ft. between ink system changes
- Auto-replenishment ink delivery for easy replacement of ink
- Built-in humidity and temperature sensor adjusts drying times for best results
- Automatic printhead alignment
- Optional take-up reel for unattended printing
- Adobe PostScript 3 as standard
- PostScript drivers for Macintosh and Windows included
- Reprints images quickly and easily from the queue
- Automatic color calibration for consistent results
- Network ready with built-in HP JetDirect EIO network card for Ethernet 10/100 Base-TX
- 4.3-GB built-in hard disk
- Print directly from your desktop PC

Not Your Father's Large Format Printer

It used to be that these large format printers were only for service bureaus — they were expensive, and took a lot of effort to use. These Entry level HPs can run as little as $2500 1700 ($2500 is more accurate for Mac solutions — below you have $1700, so we just wanted to keep it consistent.) and plug into your Ethernet network — in other words, you don't need a dedicated workstation. If you have anything above "occasional" use needs for a large format printer, this is a printer worth having on your network.

Range of Product Line

HP's smaller models (starting with the HP DesignJet 488CA printer) have Ethernet connections, but the RIP process happens in software on the Mac. HP has a full range of solutions starting at $1700 and going up to nearly $12,000 with many steps in between. To choose the model, you need to choose resolution, paper handling, software vs. hardware RIP, etc., and then the HP model will be obvious. See HP's web site for more information.

Pricing

The HP DesignJet 2500CP (the model we reviewed) series large format printer cost approximately $10k to own. This comes fully functional and fully equipped with

- Roll feed spindle, cutter knife, and holder
- HP DesignJet CP Ink System for imaging
- One each of black, cyan, magenta, and yellow
- Two rolls of selected media (paper, etc.)
- Ink storage container
- User documentation
- Drivers
- Hard disk, HP JetDirect card for Ethernet/LocalTalk, and PostScript drivers for Macintosh and Windows

HP's product line works fairly similar to the DesignJet 2500CP printer with quality of output and media/ink types being the primary difference between models. Information about HP, and its products can be found on the World Wide Web at <http://www.hp.com>.

The Roland Hi-Fi JET

The Hi-Fi JET is a proof-quality printer — the first to offer 1440 dpi resolution in a wide-format device. The Hi-Fi JET's printing system introduces a new standard for professional digital color printing. The Hi-Fi JET is calibrated to ensure accurate and consistent color reproduction of PANTONE MATCHING SYSTEM Colors, when printing in Orange and Green. The Hi-Fi JET is offers large-size proofing capability of solid PANTONE Colors using Hexachrome-licensed color technology.

Features

- True 1440x720 dpi high resolution printing
- Additional printing modes of 720, 360 and 180 dpi
- PANTONE® Hexachrome® color matching
- Two interchangeable, six-color ink set options for smooth gradations and vivid colors
- CMYK, plus Orange and Green (UV stable pigment)
- CMYK, Light Cyan, Light Magenta (dye or UV stable pigment)
- Roland ColorChoice® driver-based RIP for Windows 95/98/NT and Macintosh
- Adobe PostScript® 3
- 50 inch printing width with no length limit (40" unit available)
- Over a dozen types of media available; matte, gloss, film, backlit film, fine art papers and even canvas
- Easy set-up, as little as 30 minutes
- Ink cartridges can be changed mid-print job
- Self cleaning permanent printer heads

Additional Features

The set-up of this large format printer and included stand went so smoothly, we didn't need to look for further documentation to answer any questions. The parallel printer cable was included and the parallel card for the Macintosh installed quickly as well. All of the printer menus were very easy to understand, and provided a helpful guide covering several popular desktop publishing programs.

You can print directly from applications such as PhotoShop and Illustrator, or Quark Xpress. With the included ColorChoice® Adobe Postscript® 3 software RIP, the RIPping went very quickly. The Hi-Fi JET includes an automatic sheet cutter which will cut and separate the print job when complete. An optional take-up roller is available for rolling extra long continuous-feed prints. The permanent print heads, equipped with automatic self-cleaning functions, produce very high quality prints, photorealistic images and smooth color gradations.

Rolls or sheets of media are easily loaded and secured with a single lever. This printer offers a very wide variety of choices in printing inks, formats, sizes and media, making it extremely flexible and completely adaptable to almost any kind of print job. Loading the media was a breeze, and the front panel menus were straightforward, to the point, and extremely helpful. An optional take-up roller is available for rolling extra long continuous-feed prints. This makes unattended printing even easier.

The Roland Hi-Fi JET large format printer has a seamless user interface. Users can print right from their desktop publishing program of choice, without having to learn a new program. The RIPping program was auto-loading and called itself up when it was needed. This is a true WYSIWYG (what you see is what you get) printer. What appeared on the screen is what appeared on the output media.

Accessories

- Power cord
- Ink drain bottle
- Sheet cutter blade
- Roland ColorChoice RIP driver
- Quick Start Guide
- User's manual
- Sample File CD
- ECP (IEEE 1284) Parallel Cable
- One ink set (User specified: CMYKLcLm, CMYKOG, or CMYKLcLm Dye)
- 3 sheets sample media
- Macintosh PCI Interface Card (Optional)
- Proprietary USB to Parallel Cable (Optional)
- Optional Take Up Reel for unattended printing

This Roland printer is a high-end unit. It's meant for people that either are in the business of creating large format output, or have high quality needs (e.g., artists, professional level production, etc...). You really need to have a high-end dedicated workstation to use the product well - as the best way to hook it up is direct interface via a PCI card or USB-to-Parallel cable. The 50" printer has a retail price of around $18,000 - so you really need to have the higher end needs to justify it. But, if you do, you'll be pleased with the results.

About Roland DGA Corporation

Roland DGA Corporation is a wholly-owned subsidiary of Roland DG Corporation, a manufacturer of computer-aided input and output devices used in photography, fine art, graphic arts, sign making, engraving and 3D modeling. Introduced in fall, 1998, the Hi-Fi JET is the first wide-format 1440 dpi Piezo inkjet, and the first device with PANTONE Hexachrome technology. The six-color Hi-Fi JET provides an expanded color gamut and truer blacks, dramatically improving the realism and intensity of full-color reproduction as well as offering print permanence of over 120 years when utilizing Roland pigment inks. The Roland Hi-Fi JET is available in 50" and 40" print widths. For more information about Roland DGA Corporation and the Hi-Fi JET printers, visit their web site at <http://www.rolanddga.com>. MT

By Richard Atwell, *©2000 by Metrowerks, Inc., all rights reserved*

CodeWarrior Update

Richard Alexander David Atwell *started at Metrowerks in 1998 when he signed up to join the ranks of those who use CodeWarrior to write CodeWarrior. If you are interested in doing the same you can reach him at ratwell@metrowerks.com.*

CodeWarrior Release 5.3 Update

Just before the Christmas holiday, Metrowerks shipped its CodeWarrior Release 5.3 Update. We shipped two versions: one that contained pre-built libraries (37MB) and one that didn't (25MB) in order to make the download as small as possible for those with slower connections to the Internet.

We also updated some of our documentation (6MB) but only in PDF format. The update includes the C Compilers Reference, IDE User Guide, MSL C Reference, QuickStart Professional and Targeting Mac OS manuals. The manuals have been updated largely to document the AltiVec support in the tools.

There is also an update to the MetroNub for Java debugger plug-in. This plug-in previously shipped on the CodeWarrior Release 5 Reference CD in the pre-release folder. The MetroNub plug-in that the installer places in your Release 5 Metrowerks CodeWarrior folder is not intended to debug Java anymore. Instead, there is a separate plug-in called the SunJavaPlugin for Java targets. The release notes with the update say:

> The only feature that the old Metronub/Java debugger plug-in provides that this new plug-in doesn't is the ability to debug already running Java code without launching the application/applet that uses it from CodeWarrior. In other words, with the new Sun Java debugger plug-in, all Java debugging must originate from the IDE. Attaching to an already running VM process is not supported.

If you don't require this support there's no need to install this extra update. Visit http://www.metrowerks.com/download/ for all the latest tools. The Release 5.3 Update is planned to be the last update before CodeWarrior Release 6.

IDE 4.0.4

With the introduction of the IDE 4.0.4 you may have noticed that there are several new plug-ins in the `CodeWarrior Plugins:Debugger: folder`. These are new symbolics plug-ins and each one contains target-level symbolics support that was previously included in the IDE application itself. Of interest to Mac developers are the SymSymbolics and JavaSymbolics plug-ins.

We've made these changes so development changes to one target don't adversely effect other targets. We hope this will keep the integrated debugger working for the Mac as we make required changes for the other targets that we support.

In addition to these structural changes, the debugger settings target preferences panel has changed in the latest IDE.

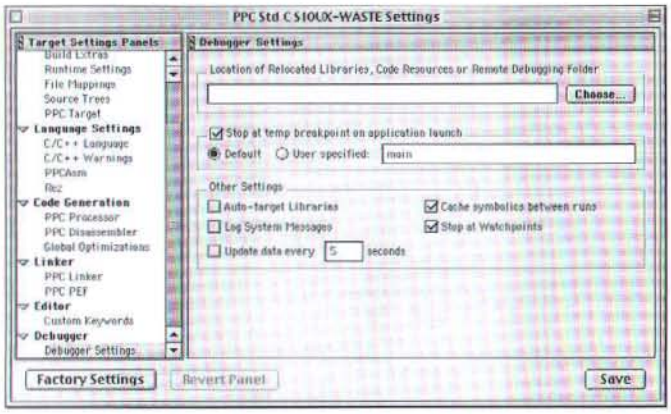


Figure 1. *new Debugger Settings Panel.*

When you open your existing projects you'll get a dialog confirmation that the project needs to be updated to the new settings. When you give the OK, each target will be updated and you'll see a project message such as:

> The data for the "Debugger Settings" settings panel in target "PPC Debug MacOS Toolbox" of project "MyProject.mcp" has been updated to the latest version.

The two new options that require updating are `Stop at temp breakpoint at application launch` and `Cache symbolics between runs`.

Instead of the old "stop at main" debugger global settings preference, we now allow a target-specific preference to let the debugger stop at the "main" as before (default behavior) or at a user specified symbol.

Simply enter a function name that must be located in the symbolics (main, __start, NewBall) and the debugger will launch your program and stop at this function instead of a hard coded symbol name. This is especially useful because it allows you to start a debugging session and execute all the way until you reach a location such as the function you are currently debugging, then halt.

Caching symbolics between runs is what the integrated debugger always used to do. Our debugger used to keep symbolics loaded until a new build was created or the project was closed. With our MPW command line tools this posed a problem if the user built them, debugged, and then attempted to build with them again without closing the project: the tools could not write some of the output files since they were still being used by the debugger.

With this option off, symbolics are discarded between runs, which frees the symbolic files so that external programs may modify them. The further benefit is that if you were to kill and restart your program, symbolics would be discarded and reloaded. That could take some time if the SYM file is large.

Compilers And MSL

We shipped MSL 5.2 and new 2.3.2 compilers with the Release 5.3 Update. There are many new language features available in the front end and improvements to the PowerPC backend.

Here are some front-end highlights:

- Support for the ISO C++ section 16.8 value for the predefined macro "__cplusplus". This is set to "199711L" to indicate conformance with the ISO C++ specification.
- Support for #pragma fullpath_prepdump on|off|reset (default off). This option can be used to show the full path of any included files in the comments in a preprocessor dump. When turned off, only the filename portion of the path is shown.
- Support for #pragma line_prepdump on|off|reset (default off). This option can be used to uncomment the "#line ..." directives in a preprocessor dump. This option it will also preserve the vertical line spacing by inserting empty lines.
- "#pragma gcc_extensions on" now allows automatic struct/array variables to be initialized with non-const values, such as function parameters. (non-ANSI C)
- #pragma warn_no_side_effect on|off| reset (default: off)
- #pragma warn_resultnotused on|off|reset (default: off)
- __option(<x>) will now return 'false' instead of an error message if <x> is an unknown option/#pragma.
- #pragma opt_strength_reduction_strict on|off|reset (default: off). This option can be used to disable strength reduction optimizations that could be unsafe because the induction variable has an unsigned type smaller than the pointer type. Previously, this option was controlled solely by the ANSI strict setting.

Some PowerPC backend highlights:

- Support #pragma function_align 4|8|16|32|64|128|reset allows specifying of the alignment of a function.
- AltiVec constants are now pooled.
- #pragma altivec_vrsave allon is now supported. It sets VRsave assuming that all AltiVec registers are in use, best used with "#pragma altivec_vrsave off" so only the parent routine updates the vrsave register.
- Added "machine 7400" directive to inline assembler

Mac OS Support

We're continually updating the support folders new that we ship CDs less often. With the Release 5.2 Update we made several changes to MacOS Support:

- Updated MacHeaders so they can be compiles with ANSI Strict ON. This usually means removing the // C++ style comments and replacing them with the /* C style comments.
- Added a stub library for the PLStringFuncs that exist within StdCLib.
- Added some common MPW files that are not part of Universal Interfaces.
- Replaced fp.h with the modified Pro4 version that fixed problems when including cmath and fp.h in the wrong order.

As always, we welcome your feedback on any subject. Contact us through our newsgroup or send us email directly using the addresses below.

- Technical Support: cw_support@metrowerks.com
- Report Bugs: cw_bug@metrowerks.com
- Suggestions: cw_suggestion@metrowerks.com

MT

By Jeff Clites <online@mactech.com>

The MP3 Phenomenon

It's nearly impossible today to pick up a magazine or newspaper and avoid seeing an article about MP3s. They're popular and surrounded by legal controversy — all the makings of a good media festival. And if you go to VersionTracker and run a search, you'll quickly find out that the Macintosh community is no more resistant to the craze than anyone else, if you didn't know this already. Hype aside, this is the sort of thing which should grab a developer's attention, because there's a big market for anything which plays, produces, converts, organizes, streams, or finds MP3s.

There's a second reason for Macintosh developers to be excited, beyond the sheer popularity, and that's QuickTime. There are three things which everyone agrees that Apple does like no other company: interfaces, multimedia, and consumer excitement. There's no question that QuickTime is right at the intersection of these strengths (despite the questionable design of the QuickTime 4 Movie Player's interface), and their quite lucrative investment in Akamai is a strong statement of their commitment to be a leader in streaming media.

On the other hand, Apple's direct pursuit of the MP3 market has been weak. QuickTime isn't the best MP3 player around, either in terms of playback quality or features (although the QuickTime 4.1 update improves this somewhat), QuickTime doesn't do MP3 encoding out of the box, and their Movie Player doesn't receive, nor does their QuickTime Streaming Server produce, MP3 audio streams in the popular SHOUTcast format. So what's going on?

There are a couple of ways to read the situation. One is that Apple is merely steering clear of the craze—its momentum is centered in the Windows community, there are legal issues with the distribution of MP3s and copyright violations, and there are licensing fees required for encoders. While this may be true, it's only half of the story. Like many of Apple's winning strategies, this one may center around a single revelation, both obvious and at the same time completely overlooked: the MP3 format is irrelevant. Not digital audio, but the idea that the consumer cares at all about the format of their files, or even what "MP3" stands for. (For the record, it's "MPEG-1 Audio Layer III," not "MPEG 3," which doesn't exist.) What they really care about is *quality*. The QuickTime Star Wars trailer taught us that—although it was available in a variety of sizes, almost everyone chose the largest, highest quality version, and you can bet that most of them were pulling these multiple megabytes painfully over a modem. The truth is that there are better audio formats than MP3 out there, both in terms of file size and sound quality. (This is of course subjective, but many have high praise for the QDesign Music 2 format, which tends to be used for the audio track of high-quality QuickTime movies.)

This is just speculation, of course, but moving forward consumers will certainly choose to listen to whatever sounds best, and this is also what artists will want to produce. Does this mean that MP3 will vanish as a format? Of course not. What it does mean is that consumers will be best served by products which are codec agnostic, and can adapt to play a variety of formats. Some of the newer MP3 hardware players are supposed to support upgrades to play future formats, and of course QuickTime has always supported multiple formats, because it's a completely modular technology; third parties can easily produce their own compressor and decompressor components. In fact I wouldn't be surprised if some day we have a portable QuickTime player, supporting video as well as audio.

So what does this all mean to the Macintosh developer today? It means that in the short term Apple has provided an opportunity for developers to step in and fill a need. In the long term, it means that the Macintosh can take the lead in the digital audio arena. If you are doing anything in the MP3 arena today, you'd be wise to use QuickTime. If you write your own encoder or decoder, implement it as a QuickTime component, and if you license one, favor one that is implemented this way. Automatically, you'll be able to support other formats that QuickTime currently plays and formats that it will play tomorrow, and you'll be able to extend its capabilities without rewriting the application itself. This will give Macintosh applications (and Windows applications which embrace Apple technologies) a unique ability to keep apace with changes in the industry.

This month I am going to leave you will a relatively small set of links to some key resources. First off is Apple's QuickTime developer site, which contains the key resources

you'll need to integrate QuickTime into your products. Second is Fraunhofer, who for better or worse holds patents to key algorithms involved in MP3 compression, and they are enforcing their rights to licensing fees. They have a web page (or a place holder for one) devoted to licensing, and you'll want to acquaint yourself with the issues involved. On a positive note, they do have arguably the highest-quality encoder available, and their place as an originator of the technology shows. They also have a few pages of introductory information about the format. Next on your list should be the MPEG Audio News, a small site which tracks news relevant to the Macintosh, and has several interesting articles, including an introduction to MPEG audio, a review of the QDesign Music codec, an encoder quality comparison, and an interview with Rafael Luebbert, the author of the historic MPecker encoder. For a look at the other formats out there, Terran's Codec Central page has information on a variety of multimedia formats, and the SoundApp site contains a page with a brief explanation of the many formats supported by this application.

Apple - Developer – QuickTime
<http://developer.apple.com/quicktime/>
Fraunhofer IIS-A - Licensing
<http://iis.fhg.de/amm/legal/>
Fraunhofer IIS-A - Layer-3 Info
<http://www.iis.fhg.de/amm/techinf/layer3/>
MPEG Audio News
<http://www.raum.com/mpeg/>
Codec Central - Info on Multimedia Codecs and Technologies
<http://www.terran.com/CodecCentral/>
SoundApp Formats
<http://www-cs-students.stanford.edu/~franke/SoundApp/formats.html>

Lastly, I would be remiss if I didn't point out icecast. It's an open source streaming audio server launched in January of 1999, and it streams MP3 audio in a format compatible with SHOUTcast, the distributed streaming audio system by Nullsoft, makers of the popular Winamp MP3 player for Windows. Since Apple's Darwin Streaming Server is open source as well, one wonders if some enterprising developer will unify the two technologies. Icecast is under the GNU Public License, so there may be a licensing mismatch there, but that doesn't prevent one from enhancing the Darwin Streaming Server to serve icecast-compatible streams without actually using code from icecast (after all, *it* was developed without the code to SHOUTcast). After all, we can't let the Linux DJs have all the fun.

icecast
<http://icecast.org>
Apple - Public Source - Darwin Streaming Server
<http://www.publicsource.apple.com//projects/streaming/>

MT

PRODUCTIVITY ENHANCER!

MacTech Magazine is where people look to find detailed and in-depth information on Macintosh technology and development. Each issue covers all the latest and most detailed information on what makes the Macintosh number one! If just one article, in just one issue, solves even one problem for you, the subscription has paid for itself!

Order Online today at www.mactech.com

P.O. Box 5200, Westlake Village, CA 91359-5200
Voice 800/MACDEV-1 • Fax 805/494-9798
Email: orders@mactech.com

ADVERTISER/ PRODUCT LIST

List of Advertisers

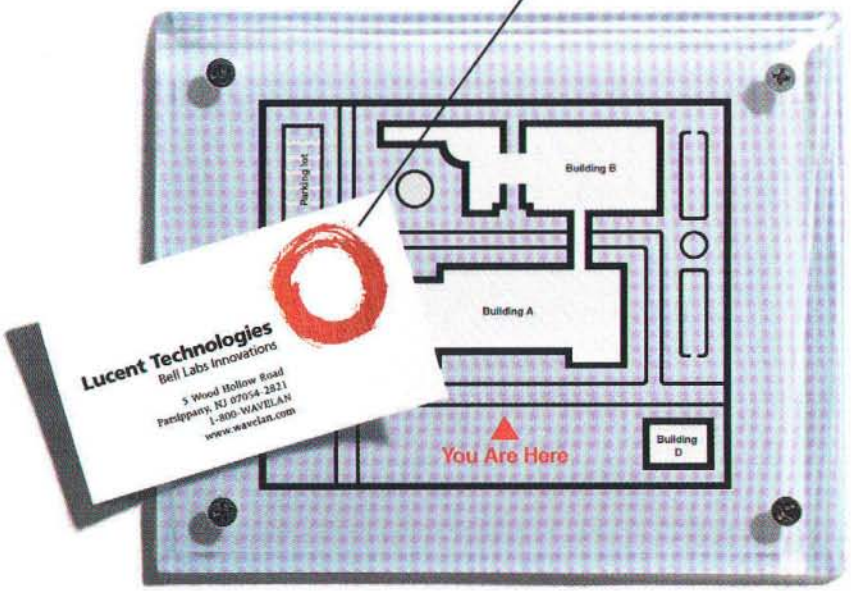
Active Concepts 21
AD Software 31
Aladdin Knowledge Systems 5
BeeHive 15
Belkin Components 37
Blueworld ..... IFC
Bowers Development 49
Developer Depot 28-29
Document Express 25
Drive Savers 61
FairCom Corporation 45
Garage.Com 58
Lucent Technologies ..... IBC
MacTech CD ROM 50
MacTech Magazine 95
Main Event 59
Mathmaesthetics 1
MDG Computer Services 63
Metrowerks ..... BC
Miller Freeman 79
New Wave Software 7
Onyx Technologies 57
OpenBase International 55
Open Door Networks 53
Panda Wave 27
Paradigma 65
Real Software 33
Rocky Mountain Ram 71
Scientific Placement 47
Smith Micro 41
Stone Tablet 39
Technosoft 13
UNI SOFTWARE PLUS 9
Water's Edge 19

List of Products

ADB Device • BeeHive 15
AppMaker • Bowers Development 49
BugLink • PandaWave 27
C++ Developers Kit • Technosoft 13
c-tree Plus • FairCom Corporation 45
CodeWarrior • Metrowerks ..... BC
Conference • Garage.Com 58
Conference • Miller/Freeman 79
Data Recovery • Drive Savers 61
Development Tools • Developer Depot 28-29
E-Mail and Webform Management Tool • Document Express 25
Funnel Web • Active Concepts 21
Hardware • Belkin Components 37
Job Placement • Scientific Placement 47
Lasso Web Data Engine • Blueworld ..... IFC
MacHasp USB • Aladdin Knowledge Systems 5
Magazine • MacTech Magazine 95
Memory • Rocky Mountain Ram 71
OO File • AD Software 31
REALbasic • Real Software 33
Reference • MacTech CD ROM 50
Relational Database Engine • OpenBase International 55
Resorcerer • Mathmaesthetics 1
Scripter • Main Event 59
ShareWay IP • Open Door Networks 53
Software Piracy Intervention • New Wave Software 7
Software Solutions • Smith Micro 41
Spotlight • Onyx Technologies 57
Stone Table • Stone Table Publishing 39
Valentina • Paradigma 65
VOODOO Server • UNI SOFTWARE PLUS 9
WaveLAN • Lucent Technologies ..... IBC
WebServer 4D • MDG Computer Services 63

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

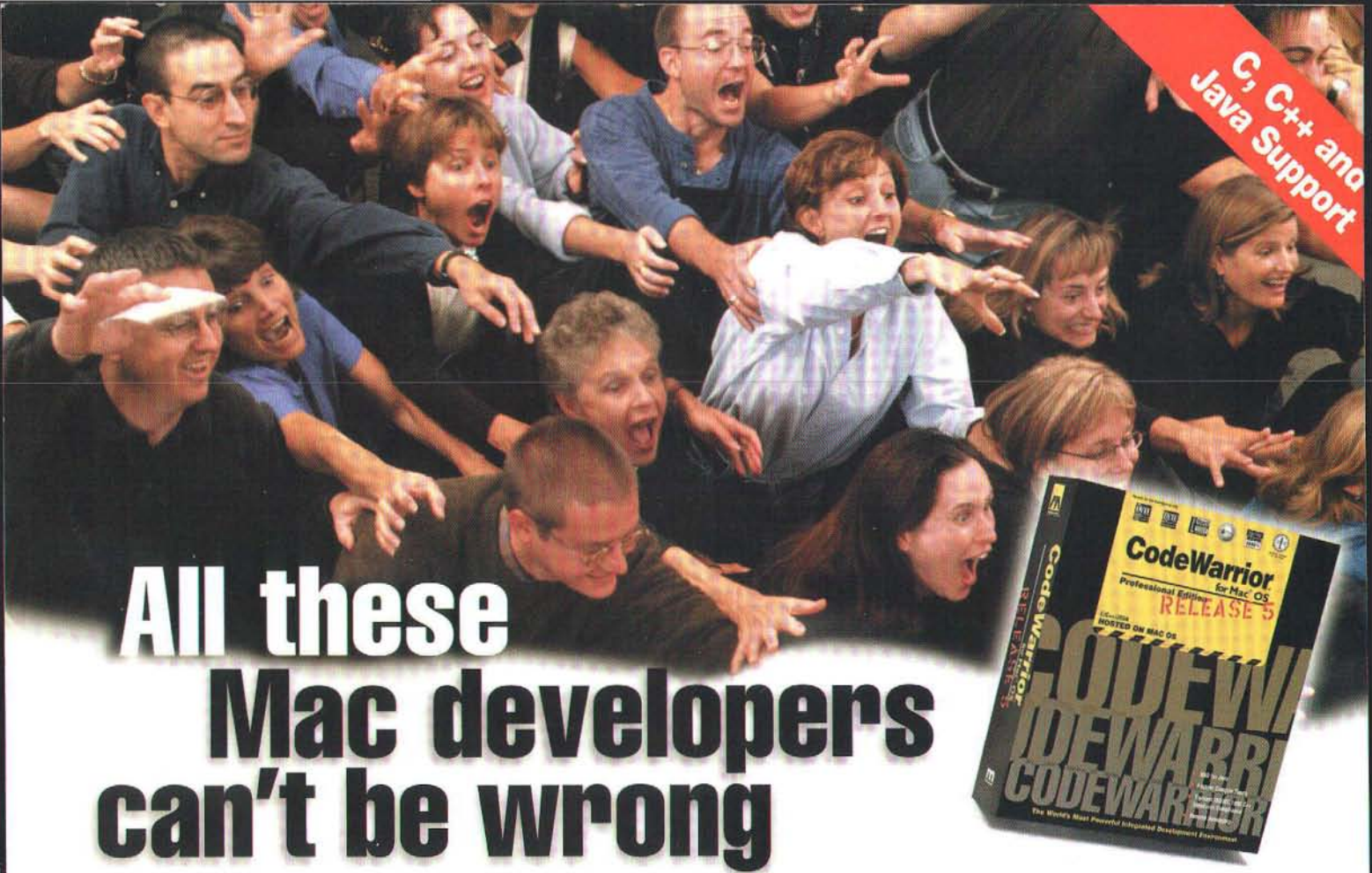
You are here
(parking lot
water cooler
conference room
huddle room
boss' office
big boss' office
latte bar)
on your laptop,
on your
wireless LAN



WaveLAN® Wireless Solutions from Lucent free everyone in your company from the deskbound, officebound network.

With WaveLAN/IEEE Turbo 11Mb Cards, people can stay connected, and work productively at the speed of Ethernet anywhere they meet, greet or get inspired. Standards-based WaveLAN makes wireless LANs truly efficient, fast and affordable. See how you can boost your productivity at 1-800-WAVELAN or www.wavelan.com.

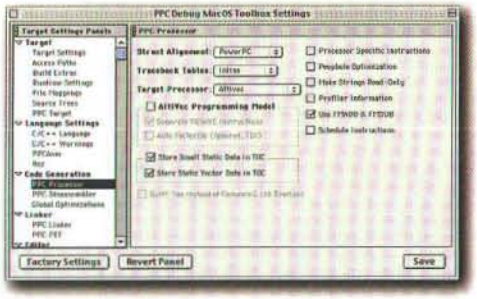
We make the things that
make communications work.™



All these Mac developers can't be wrong

CodeWarrior is the only serious choice for professional Mac® development.
Chances are, every Mac program you use was created using CodeWarrior. In fact, more than 90% of all commercial Mac OS applications are built with CodeWarrior.

Built with the Mac developer in mind.
The CodeWarrior Integrated Development Environment (IDE) has won every major award given to Mac development software. It's the worldwide standard for creating Mac software, and continues to lead the industry.



CodeWarrior keeps you ahead of the curve with AltiVec and future OS X support.

Check out these CodeWarrior benefits:

- ✔ The state-of-the-art IDE ***shortens your development time***.
- ✔ ***Everything you need in one box...*** project manager, editor, compilers, debuggers, linkers, assemblers, profilers, a file compare utility, our world-class framework PowerPlant™, and the flexibility to plug-in third party components.
- ✔ Use ***the same familiar toolset*** to develop for all the major platforms, including Windows®, Linux, Solaris™, game consoles and embedded platforms.
- ✔ Tightest ISO/IEC ***C++ standards compliance***.
- ✔ ***C, C++ and Java™ support***.

Go with the world-class standard.
Use the same powerful, flexible, fast toolset that built world-class apps like Adobe® Photoshop® and Netscape® Navigator®.

See for yourself.
Don't just take our word for it. Get your FREE* 30-day evaluation edition of CodeWarrior for Mac OS and put it to the test.

FREE!* 30-day Eval

Discover why over 200,000 programmers worldwide use CodeWarrior.

Call today or order online FREE 30-day evaluation

www.metrowerks.com/go/mactech

1-800-377-5416

*Non-refundable shipping and handling of $1.95 per CD applies, plus state sales tax if applicable.

Hurry! This offer expires 4/30/2000.

metrowerks®

CodeWarrior® for Mac® OS

Java Report BYTE MacUser Macworld

© 1999-2000 Metrowerks. All rights reserved. Metrowerks, the Metrowerks logo and CodeWarrior are registered trademarks, and PowerPlant is a trademark of Metrowerks Inc. All other products are trademarks of their respective companies. All prices are in US dollars; offer good in US and Canada only. CodeWarrior for Mac OS, Professional Edition, targets Mac OS and Windows 95/98/NT only; other targets available separately.